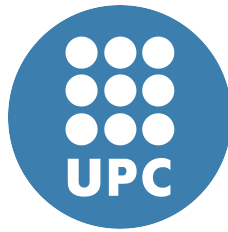


Development and Certification of Mixed-criticality Embedded Systems based on Probabilistic Timing Analysis



Irune Agirre Troncoso
Computer Architecture Department
Universitat Politècnica de Catalunya

A thesis submitted for the degree of
PhD in Computer Architecture

May 2018

Development and Certification of Mixed-criticality Embedded Systems based on Probabilistic Timing Analysis

Irune Agirre Troncoso

May 2018

Universitat Politècnica de Catalunya
Computer Architecture Department

Thesis submitted for the degree of
Doctor of Philosophy in Computer Architecture

Advisor: Francisco J. Cazorla,
Barcelona Supercomputing Center and IIIA-CSIC
Co-advisor: Mikel Azkarate-askatsua,
IK4-Ikerlan Technology Research Center

The work reported in this Thesis has been conducted in collaboration between the Universitat Politècnica de Catalunya, the Dependable Embedded Systems department of IK4-Ikerlan Technology Research Center and the Computer Architecture and Operating Systems (CAOS) group of the Barcelona Supercomputing Center (BSC).

Abstract

An increasing variety of emerging systems relentlessly replace or augment the functionality of mechanical subsystems with embedded electronics. For quantity, complexity, and use, the safety of such subsystems, often subject to regulation, is an increasingly important matter. Those systems are subject to safety certification to demonstrate system's safety by rigorous development processes and hardware/software constraints. The massive augment in embedded processors' complexity renders the arduous certification task significantly harder to achieve. The focus of this thesis is to address the certification challenges in multicore architectures: despite their potential to integrate several applications on a single platform, their inherent complexity imperils their timing predictability and certification. For the former, Measurement-Based Probabilistic Timing Analysis (MBPTA) has recently emerged as an alternative to deal with hardware/software complexity. The innovation that MBPTA brings about is, however, a major step from current certification procedures and standards.

The particular contributions of this Thesis include: (i) the definition of certification arguments for mixed-criticality integration upon multicore processors. In particular we propose a set of safety mechanisms and procedures as required to comply with functional safety standards. For timing predictability, (ii) we present a quantitative approach to assess the likelihood of execution-time exceedance events with respect to the risk reduction requirements on safety standards. To this end, we build upon the MBPTA approach and we present the design of a safety-related source of randomization (SoR), that plays a key role in the platform-level randomization needed by MBPTA. And (iii) we evaluate current certification guidance with respect to emerging high performance design trends like caches.

Overall, this Thesis pushes the certification limits in the use of multicore and MBPTA technology in Critical Real-Time Embedded Systems (CRTES) and paves the way towards their adoption in industry.

Acknowledgements

I would like to extend thanks to the many people who so generously contributed to the success of this thesis.

First and foremost I would like to express my deepest gratitude to my advisors: Dr. Francisco J. Cazorla and Dr. Mikel Azkarate-askatsua for giving me the opportunity to develop this thesis under their direction. I am grateful for their endless support, time, encouragement and knowledge that guided me towards the objectives of this thesis.

During these years I had the privilege of collaborating with excellent professionals. I would like to acknowledge Dr. Jaume Abella for sharing his expertise so willingly during my stays in the Barcelona Supercomputing Center and Dr. Carles Hernandez, Dr. Enrico Mezzetti and prof. Tullio Vardanega for their constructive contributions to this thesis. I would also like to thank all members with who I had the pleasure to collaborate in the European research project PROXIMA, which laid the foundations of MBPTA for multicore architectures.

Many thanks to IK4-Ikerlan for the opportunity and funding that made this thesis possible. I feel lucky to be sharing my workspace with a great group of friends: Irune, Iñaki I., Itxaso, Ane, Peio, Imanol A., Charly, Unai U., Josu L., Asier and Iban A. Special thanks to those colleagues that played a part in this thesis: Itxaso Saenz, Asier Larrucea, Carlos F. Nicolas and Jon Perez. It has also been a great honor to visit the CAOS group in the BSC. Many thanks to all its members for their hospitality.

Finally, I would like to thank my family and friends for their unconditional love and support. To all those friends that during these years were by my side: Ibane, Maitane, Erik, Raquel, Miren, Leire, Gloria and Maore. And specially to my fellow travelers in this adventure: Maria and Aitor, for being there for me when I needed it the most.

Eskerrik beroenak azkenik senitartekoei, bereziki ama, aita, ahizpa eta Asier-i, emandako aholku, babes eta maitasunagatik. Baita Danel-i ere, azken urtean etxean zabaldu duen alaitasunagatik.

Contents

1	Introduction	1
1.1	Trends in Mixed-criticality Systems	2
1.1.1	Increased Integration Needs	3
1.1.2	Higher Performance Requirements	4
1.2	Future Mixed-criticality Systems' Challenges	4
1.2.1	Certification	5
1.2.2	Timing Predictability	5
1.3	Contributions	7
1.3.1	Multicore Mixed-criticality Certification	8
1.3.2	MBPTA Potential Adherence to Standards	9
1.3.3	Practical Multicore Certification Compliance	9
1.4	Thesis Organization	10
1.5	List of Publications	11
2	Background and Basic Concepts	15
2.1	Basic Concepts	15
2.1.1	Safety-critical Embedded Systems	15
2.1.2	Multicore Processors	20
2.1.3	Timing Analysis	22
2.2	Mixed-criticality Integration on Multicores	27
2.2.1	Partitioning on Multicore Processors	29
2.2.2	Certification	31
2.3	Timing Analysis on Multicores	32
2.3.1	Minimizing Multicore Contention	32
2.3.2	Accounting for Multicore Contention	33
2.3.3	Certification	33
2.4	Summary and Conclusions	34
3	Experimental Setup	37
3.1	Multicore Architectures	38
3.1.1	COTS Multicore: P4080	38

3.1.2	Ad-hoc Multicore: LEON3-MC	39
3.1.3	Certification-friendly Multicore: AURIX TC27x	39
3.2	Applications	40
3.2.1	Industrial Railway Application	40
3.2.2	Industrial Automotive Application	42
3.2.3	Microbenchmarks	43
3.3	Timing Analysis: MBPTA	44
3.3.1	MBPTA-compliant Time Randomized Platforms	44
3.3.2	Collecting Measurements	46
3.3.3	Probabilistic Analysis	46
4	Safety Argument for Multicore Mixed-criticality Systems	47
4.1	Notation	48
4.2	Overall Safety Argument	49
4.3	Common-practice Federated Approach	51
4.4	Mixed-criticality Integration	52
4.4.1	Integrated Safety Architecture	53
4.4.2	Failure Analysis	55
4.4.3	Safety Measures	58
4.5	Multicore Integration	63
4.5.1	Multicore Safety Architecture	63
4.5.2	Independence in Multicore Architectures	64
4.6	Summary	66
5	Multicore Mixed-criticality Safety Concept	67
5.1	Glossary	68
5.2	Railway Safety Concept	69
5.2.1	Safety Requirements	69
5.2.2	Federated Safety Concept	70
5.2.3	Integrated Node Architecture	73
5.2.4	Failure Analysis	76
5.2.5	Safety Measures	78
5.2.6	Detailed Tables	85
5.3	Automotive Safety Concept	90
5.3.1	Safety Requirements	90
5.3.2	Federated Safety Concept	91
5.3.3	Integrated Node Architecture	94
5.3.4	Failure Analysis	95
5.3.5	Safety Measures	97
5.3.6	Detailed Tables	106
5.4	Summary	110

6	Fitting Execution-Time Exceedance into Safety Standards	113
6.1	Systematic and Random Faults in IEC 61508	114
6.1.1	Software Faults	115
6.1.2	Hardware Faults	116
6.2	Execution Time Exceedance Rates	117
6.3	MBPTA Adherence to IEC 61508	119
6.3.1	MBPTA Application and Feasibility	119
6.3.2	MBPTA in IEC 61508 Safety Life-cycle	122
6.3.3	Cut-off Probability Determination	124
6.4	Domain-specific Standards	125
6.5	Experimental Support Evidence	126
6.5.1	Collecting Analysis-time Observations	126
6.5.2	Application of MBPTA	128
6.6	Summary	132
7	Safety-Related Pseudo-Random Number Generator	133
7.1	High-Quality Low-Cost PRNG	134
7.1.1	Sought Properties	135
7.1.2	MBPTA Convenient PRNG	136
7.2	PRNG for Multicore Platforms	139
7.2.1	Multicore Requirements	139
7.2.2	Sharing PRNG Modules	140
7.3	SIL 3 Compliant PRNG	141
7.3.1	Safety Requirements	143
7.3.2	Safety Techniques	143
7.3.3	System Reaction to Errors	145
7.4	Evaluation	146
7.4.1	LFSR's Sought Properties	146
7.4.2	PRNG Integration in a Multicore Prototype	147
7.5	Summary	148
8	Practical CAST-32A Compliance on COTS Multicores	151
8.1	Overview of CAST-32A Principles	152
8.1.1	General Definitions	152
8.1.2	CAST-32A Objectives	153
8.2	Interpretation and Achievability of CAST-32A Objectives	154
8.2.1	Robust Resource Partitioning (RRP)	154
8.2.2	Robust Time Partitioning (RTP)	157
8.2.3	Individual Objectives	158
8.3	Evaluation on Complex COTS Multicore	159
8.3.1	Defining Hardware Configuration Settings	159

8.3.2	Identifying Interference Channels	162
8.4	Summary	167
9	Conclusions and Future Work	169
9.1	Summary of Contributions	169
9.2	Impact	171
9.3	Future Work	173

List of Figures

1.1	Logical organization of Thesis' contributions.	8
2.1	Relation among safety certification standards.	16
2.2	IEC 61508 safety life-cycle (IEC61508).	17
2.3	Illustrative example of (timing) Fault, Error and Failure.	19
2.4	Generic COTS based multicore architecture.	20
2.5	Basic concepts for Worst-Case Execution Time (WCET) analysis (WEE+08).	23
2.6	Classification of timing analysis techniques (AHQ+15).	24
2.7	MBPTA concepts and example pWCET outcome.	25
3.1	Thesis contribution and evaluation dependencies.	37
3.2	High level block diagram of the P4080 (memory hierarchy).	38
3.3	High level block diagram of the LEON3-MC (memory hierarchy).	39
3.4	High level block diagram of the AURIX TC27x (memory hierarchy).	40
3.5	Block diagram of the railway case-study application.	41
3.6	Block diagram of the automotive case-study application.	43
3.7	Execution time collection for MBPTA.	46
4.1	Principal Goal Structuring Notation (GSN) elements (GSN11).	48
4.2	Summary of goals in the safety argumentation.	49
4.3	Top level argument structure for the mixed-criticality integration based on a certified federated system.	50
4.4	Common-practice federated system certification - Goal G2 (M1.1).	52
4.5	Mixed-criticality integration argument - Goal G3 (M1.2).	53
4.6	Safety architecture for partitioned processor (MxCPU).	54
4.7	Argument for the avoidance of systematic faults - Goal G4 (M3.1).	58
4.8	Argument for the co-existence of mixed-criticality applications - Goal G5 (M3.2).	60
4.9	Argument for the runtime control of errors - Goal G6 (M3.3).	61
4.10	Safety chain for safe state activation.	63

4.11	Reference safety architecture for generic multicore (MxCPU).	64
4.12	Multicore temporal independence argument - Goal G7 (M5.1.2).	65
5.1	Railway case study context diagram.	70
5.2	Safety concept (one processor; partitioned).	74
5.3	Safety concept in partitioned multicore processor.	75
5.4	Safety chain for ETCS safe state activation.	82
5.5	MBPTA application in the railway case study.	85
5.6	Federated system architecture for the automotive use case.	91
5.7	Automotive safety concept (one processor, partitioned).	94
5.8	Safety concept in AURIX TC27X multicore processor	95
5.9	Safety chain for CC safe state activation.	102
5.10	Memory and time segregation on core 0.	102
5.11	Platform protection mechanisms against spatial interferences.	103
5.12	MBPTA application in the automotive case study.	105
6.1	Schematic view of IEC 61508 treatment of systematic and random faults in the life-cycle.	115
6.2	Implications of SSR on a typical development cycle.	122
6.3	Sketch of how MBPTA fits in IEC 61508 software development process.	123
6.4	Observed execution time variability in the railway application.	127
6.5	Execution time variability caused by program layout (Path #7).	128
6.6	The pWCET distributions computed by MBPTA for the railway application subset path 7 (longest) and 9 (shortest).	129
7.1	Proposed implementation of the MWC PRNG.	137
7.2	168-bit LFSR implementation (producing 1 & 32 bits).	138
7.3	PRNG integration in the LEON3-MC architecture.	142
7.4	Safety architecture for the PRNG modules.	144
7.5	CDF for 1000 runs of the matrix benchmark.	148
8.1	Examples of P4080 hardware setups.	161
8.2	Impact of core local contenders on <i>rsk</i> , <i>DL1lh</i> , <i>L2lh</i> , <i>L3lh</i> and <i>mem</i> .	163
8.3	DL1 Hit latency variation across cores and runs.	164
8.4	Impact of coherence on execution time in isolation.	165
8.5	L3 contention under different HWS.	166

List of Tables

2.1	Criticality levels on different domain standards (Bou15).	18
4.1	Simplified Failure Mode and Effect Analysis (FMEA) for partitions.	56
5.1	Main safety requirements for the EVC.	69
5.2	Summary of safety measures and IEC 61508 to EN 5012X references.	79
5.3	pWCET at relevant exceedance thresholds for railway case study. .	84
5.4	Simplified Failure Mode and Effect Analysis for railway case study (partitions).	86
5.5	Railway system measures for runtime error control.	88
5.6	Railway system reaction to errors.	89
5.7	Main safety requirements of the cruise control system.	90
5.8	Summary of safety measures and IEC 61508 to ISO 26262 references.	98
5.9	pWCET at relevant exceedance thresholds for automotive case study.	105
5.10	Simplified Failure Mode and Effect Analysis for automotive case study (partitions).	106
5.11	Automotive system measures for runtime error control.	108
5.12	Automotive system reaction to errors.	110
6.1	IEC 61508 quantification metrics (IEC61508).	117
6.2	pWCET bounds at relevant exceedance thresholds (in processor cy- cles).	130
7.1	Random bit requirements in reference multicore and LEON3-MC. .	140
7.2	Safety measures and techniques in the PRNG.	144
7.3	System reaction to errors.	145
8.1	Summary of CAST-32A objectives (CAST-32A).	153

Chapter 1

Introduction

In the last decades, the ever-increasing expansion and evolution of computing systems have considerably increased their prevalence in everyday life activities. While the most popularly recognized use of computing systems is that of desktop computers or servers, the vast majority of them are embedded in a wide variety of quotidian appliances up to the point that it would not be far-fetched to state that all modern technology areas lean on *embedded computing systems* (LS17). Embedded systems appear in applications as diverse as simple alarm clocks, medical devices, multimedia products, industrial machinery and transportation systems among others. This unprecedented growth is motivated by their potential to accomplish complex tasks that replace systems' components traditionally committed by mechanical or electrical systems and their ability to integrate new functionalities that improve the automation, and hence ease of use, of systems (CVH08; Kop11; LS17).

As a consequence of the aforementioned ubiquity of embedded systems, they are often used in critical domains where a system failure can have destructive effects such as the loss of lives or devastating environmental damage (*safety-critical*), substantial economic and property losses in an organization (*business-critical*), disruption of critical activities adversely affecting a company or society (*mission-critical*) or loss of sensitive personal data (*security-critical*). Commonly, the correctness of these critical embedded systems does not only depend on the functional results of the computations, but also on their timing response in which case they are accordingly categorized as *Critical Real-Time Embedded Systems (CRTES)*. In CRTES, the temporal behavior is – at least – as important as their functional behavior as missing preset time boundaries, called *deadlines*, can lead to severe consequences. In these circumstances, it is imperative to demonstrate that CRTES comply with the legal directives in place in each state or country before they are allowed to be deployed for operation. This involves undergoing a *certification* process where an independent person, often from an independent certification entity or in-house department, must approve that the system is suitable and safe enough

for its intended use.

To aid in the certification process, there is a set of standards that dictates the needs of the critical system and establishes a number of constraints throughout its development cycle (e.g., EN 50126 in railway and ISO 26262 in automotive). Depending on the risk associated with the consequences, a discrete criticality attribute is assigned to each CRTES subsystem, known as safety integrity, assurance or criticality level. This criticality level highly constrains the system design as special considerations must be applied to guarantee the correct operation of the system. As a rule of thumb, the higher the criticality attribute, the higher is the risk against which the system must be protected, and consequently, the more stringent the requirements imposed by the standards. With each cycle of innovation, CRTES are confronted with increasing demands to support new functionality resulting in a high number of interacting functions, often with differing integrity requirements (*mixed-criticality*). Taking as example the classical automotive case, 80% of innovation in a car is driven by electronics (Far14; Nel10). While a contemporary car already contained dozens of hierarchically interconnected Electronic Control Unit (ECU)s in 2005, in the last decade this number has increased up to few hundred by integrating advanced functionalities like infotainment, telematics, emissions control and many safety-related applications like drive-by-wire, Anti-Lock Braking System (ABS) or cruise control system (CC) (But12; Far14). This trend is expected to continue in the future with the advent of autonomous driving cars (Reb17) and it is estimated that electronics will take the larger share of the total car costs (Nel10).

With large number of interconnected subsystems of differing criticality levels, standards pose additional requirements to ensure that the applications that comprise the mixed-criticality system cannot adversely affect to each other and allow, by ensuring *independence of execution*, the *incremental certification* of each application according to its own assigned integrity level (IEC61508). Incremental certification considerably simplifies the development and certification process as otherwise, all components that share the same computing platform shall be certified according to the highest integrity level present on the system what is effort and cost-prohibitive (WT08; PGN+14).

1.1 Trends in Mixed-criticality Systems

CRTES have come a long way since their beginning. This section gives an insight of the current trends in CRTES with emphasis on their increasing integration and computing demands.

1.1.1 Increased Integration Needs

In the past, traditional CRTES architectures have followed a *federated architecture paradigm*: CRTES were composed of a number of functionally independent interconnected subsystems with critical functions implemented on a number of dedicated subsystems, physically separated from each other. This physical separation allows incremental certification by construction, provided that the *field bus* guarantees independence of execution among the interconnected subsystems. However, the increasing functional complexity in the real-time industry challenges the continued viability of this architecture paradigm (Kop08). In the federated design, the addition of new functionalities requires adding new computing subsystems that are restricted by the limited scalability, flexibility or extensibility. Dealing with increasing number of processors and associated communication networks leads to bigger reliability challenges (e.g., Electromagnetic Compatibility sensitivity, increased number of connector and cables, etc.) and to bandwidth and quality of service limitations. Altogether, federated designs require high maintenance and upgrade costs. This is reflected in a wide variety of industry sectors where transportation systems, such as automotive or railway, prevail. In such systems, the increasing number of ECUs, cables, electrical parts and connectors decrease the overall reliability of the system and raise the cost-size-weight factor (PGN+14). For example, between 30-60% of electrical failures in the automotive domain are attributed to connector problems (SMM00) and in the avionics domain, having dedicated computing systems for each software application results in hundreds of kilometers of cabling per aircraft (Iti07). This has caused that several modern implementations already adopt *integrated* approaches where several applications, frequently exhibiting different criticality characteristics, run together on a single hardware platform (Kop04).

The paradigm shift from federated architectures towards integrated ones builds on the principle of sharing same platform resources by multiple system functions based on the concept of *partitioning*. Partitions provide separation among the applications and design fault containment to prevent any partitioned application from causing a failure in another partitioned application (AMI+11). A robust partitioning technique therefore guarantees separation to allow the *incremental certification* of mixed-criticality systems in an equivalent manner to the federated approach (ARINC653). The avionics domain has been successfully integrating distributed avionics subsystems in centralized processing units under the umbrella of *Integrated Modular Avionics (IMA)* (Ram07; DO297; SAE01). The significant benefits witnessed by the avionics domain thanks to IMA has also motivated other domains to adopt equivalent approaches like IMA for space (Win12) or the Automotive Open System Architecture (AUTOSAR) in the automotive field (AUTOSAR; DS10). These benefits include improved modularity and soft-

ware re-use, weight and cost reduction and decreasing up to half the amount of processor units in an airplane (Ram07). Following the trend towards increased integration, the next natural step is considering parallel processing architectures such as multicores to deal with the increasing performance requirements further explained in next subsection.

1.1.2 Higher Performance Requirements

Besides the increasing number of processing units, the continuous technological progress that CRTES are experiencing come along with increasingly complex software and hardware systems (PGT+14). Taking lines of code as representative proxy metric for complexity and performance requirements, the amount of lines of code within a system is experiencing an exponential growth (Cha09; But12). This gets exacerbated as new complex functionalities, like autonomous driving technology, continue to be integrated. As a result, increased computing performance requirements cause the traditionally used single-core processors reach their physical performance, complexity, power consumption and heat dissipation limits.

As an alternative to the infeasibility of further increasing clock speeds and the limits of Moore's law on single-cores (Cor12), CRTES industries have started to transition to multicore processors. In theory, multicore processors are an attractive solution for CRTES as they offer increased performance rates with reduced cost, size, weight and energy consumption (PGN+14; PGT+14; But12; BCD+15). Furthermore, their high performance interestingly makes them a key enabling technology for integrating multiple mixed-criticality applications on a single silicon die.

These advantages encourage even the most conservative CRTES domains to venture considering multicore solutions (PDK+15; BCS+16). Furthermore, the use of readily available and pre-tested *Commercial Off-The-Shelf (COTS)* components is on the rise, as they considerably shorten the development time and associated costs (TLH97).

1.2 Future Mixed-criticality Systems' Challenges

In spite of the aforementioned trends that promote the inclusion of multicore processors for mixed-criticality integration, several challenges related to the certification and timing predictability of multicore approaches must be considered before endorsing multicores' unrestrained adoption in CRTES.

1.2.1 Certification

Unlike conventional computing systems, where multicores are well established, CRTES are bound to be certified. Certification is the process of proving (providing evidence) that the design of a product complies with what it is established in the relevant standards. This is a fundamental step to support a declaration of conformance with legal product directives, which is compulsory to allow their deployment and operation. Certification requires a robust system design and development process that involves high efforts even for single-core processors. As a consequence, the CRTES industry has adopted the conservative trend of employing simple, predictable and proven-in-use processors. As an illustrative example, the automotive industry is still using 8-bit architectures on some of the ECUs that host safety-related software ([Far14](#)). On the other hand, modern multicore platforms provide rather the opposite properties with complex high performance features, lack of temporal predictability, weak support in the established standards and absence of history of use in the CRTES domain.

Regarding mixed-criticality, although existing solutions have succeeded on single-cores ([Kop04](#); [Ram07](#); [DS10](#)), the integration solutions adopted for single-core processors do not scale well (in terms of certification) on multicore architectures. New issues arise from their inherent complexity and unpredictability that challenge having sufficient guarantees of independence of execution among functions required to conduct an incremental certification. Resource sharing and high-performance hardware features like caches are determinant in providing the increased multicore performance. However, these features considerably complicate platform predictability and hence, determining CRTES' worst-case performance. In this scenario, a trade-off between increased average performance and separation for mixed-criticality is often necessary. Several emerging hardware designs favor time predictability and ease certification ([PQC+09](#); [SEH+12](#)). However, COTS multicore architectures incorporate such features very slowly. In spite of the opportunities that COTS processors provide for embedded system designers, their adoption in CRTES is not straightforward as they are primarily designed to increase average performance at the cost of increasing complexity. Fortunately, in the last years, certification-amenable processors are being slowly introduced in the market, where manufacturers – most of them targeting the automotive domain – start to integrate principles such as, fault management and safety measures in their commercial architectures ([TMS570](#); [MCP5xx](#); [AURIX](#); [ZynqFS](#)).

1.2.2 Timing Predictability

Standards impose rigorous analysis and verification processes. Timing is an important aspect for the verification and validation of CRTES to ensure deadlines

are met and confirm the temporal schedulability of the system. In general, the verification and validation of embedded systems require high effort, resources, and time. Subsequently, verification and validation activities are considered as the most costly development phases; more notably when safety or availability issues are involved (DGG09). Despite strict testing procedures are well established to verify conformance to functional specifications, generally temporal properties are not equally verified. The quality of the timing verification, i.e., evidence that the software is timely correct with respect to its specification, is predominantly dependent on previous experience and engineering judgment (CVH08). In this line, a common industry practice is running a set of tests and measuring their execution time for then adding a conservative *safety margin*, typically 20% of the maximum observed value (also known as High-Water Mark (HWM)). While the intend of the safety margin is to compensate for the uncertainties of the testing process, complex hardware complicates the selection of a suitable margin and it is almost never proven that the resulting system will meet its temporal requirements under every possible hardware and software condition that it may encounter at operation.

The intrinsic complexity of multicore architectures further exacerbates this issue and negatively impacts the existing timing analysis solutions. Two main timing analysis paradigms have been used in industrial domains so far: Static Timing Analysis (STA) and Measurement-Based Timing Analysis (MBTA) (WEE+08). In STA approaches, an estimate of the WCET is obtained by analyzing the program instead of executing it. Therefore, these techniques require full characterization of the execution behavior at all levels of the system, including abstract models of the hardware. When applied to complex systems, such as multicores, it is very hard to statically predict all runtime dependencies (AHQ+15). Assuming worst-case scenarios for all uncertainties incurs high pessimism. MBTA techniques, on the contrary, collect execution time measurements of the program of interest when it is executed on the real hardware or a simulator. As explained above, this is the most common industry practice (WEE+08) and, in this case, the challenge rests on guaranteeing that the worst-case (or a good approximation to it) is encountered during the test campaign. In fact, analyzing the worst-possible execution conditions involves that low-level architectural features, which can contribute to significant execution time variations, are exercised in the measurement runs taken during the timing analysis process. Creating the test cases that factor in all possible combinations is not realistically possible given the inordinately large input space that software tasks may have and the huge number of potential states of complex hardware. Accordingly, timing analysis techniques are required to relieve the user from the high burden of analyzing and controlling all low level hardware-software interactions and to obtain safe WCET estimates without incurring undue pessimism. Not surprisingly therefore, no golden solution exists so far for the estimation of

1.3 Contributions

trustworthy timing guarantees on modern multicore processors ([AHQ+15](#)).

Recently, a novel promising approach based on theory of probabilities has emerged, Probabilistic Timing Analysis (PTA) ([CSH+12](#); [CQV+13](#); [HHM09](#)). PTA in general and its measurement-based variant called Measurement-Based Probabilistic Timing Analysis (MBPTA) in particular, derive WCET estimates with an associated probability of exceedance. Accordingly, as opposed to the traditional timing analysis approaches that compute a single WCET estimate value, PTA derives a probabilistic WCET distribution called probabilistic WCET (pWCET). Further explained in the Background Chapter (Section [2.1.3](#)), MBPTA has the potential to replace the qualitative safety margin selection (based on engineers' experience) by a quantitative justification based on its probability of exceedance. To this end, MBPTA rests on the premise that the timing behavior of certain processor resources is time randomized or upper-bounded ([KQA+14](#)). This practice serves to deal with uncertainties and notably reduces the need for user-provided information during timing analysis. MBPTA has shown to be competitive with state-of-the-art timing analysis techniques for single-core architectures ([CQV+13](#); [WKL+13](#)) and has been recently ported to multicore processors with promising results to handle the complexity of high-performance hardware features like caches ([CAA+16](#); [KQA+16](#); [WKG+15](#)). However, due to its novelty, multicore MBPTA technology has not been yet evaluated with regard to CRTES certification.

1.3 Contributions

The challenging certification and time predictability of multicores are a pre-requisite for mixed-criticality integration and hence, the main motivating factors for this Thesis. Subsequently, this Thesis proposes mixed-criticality design and development considerations for multicore processors compliant with safety standards in order to ease their adoption in the safety-critical CRTES domain. Further, this Thesis pursues MBPTA's industry-readiness. To this end, it elaborates on the safety implications of MBPTA as a means to verify timing specifications and to provide sufficient guarantees of independence of execution among mixed-criticality safety functions in multicore processors. Finally, the ultimate goal of this Thesis is to assess the practical compliance of current COTS multicore design trends to existing multicore certification guidance, paying special attention on timing. Accordingly, the contributions of this Thesis are classified into three major axes (themes) as sketched in Figure [1.1](#): (i) mixed-criticality certification upon multicores; (ii) MBPTA considerations to fit in safety standards; and (iii) practical multicore certification compliance. The contributions to each theme are structured in chapters. Next subsections summarize both themes and contributions.

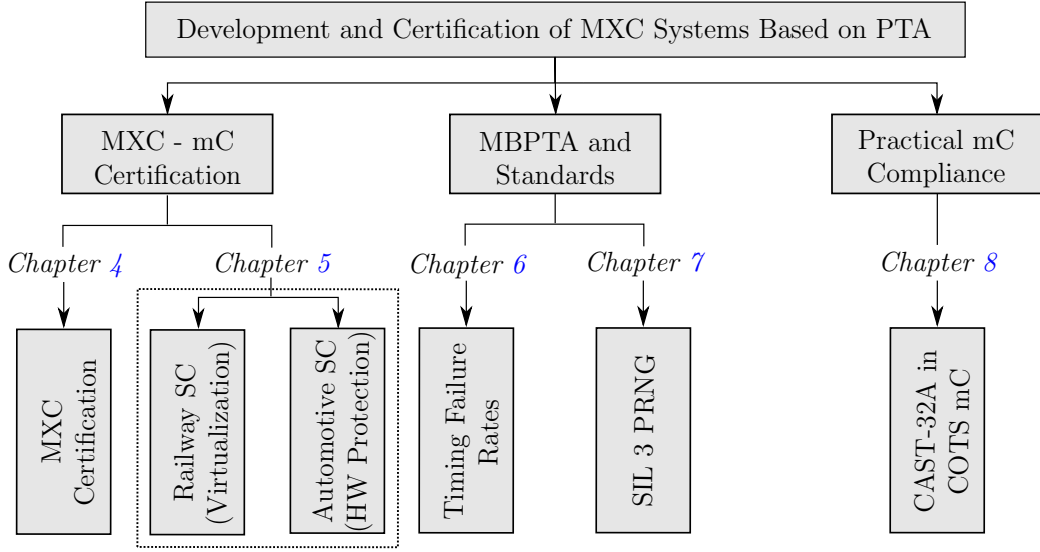


Figure 1.1: Logical organization of Thesis' contributions.

1.3.1 Multicore Mixed-criticality Certification

- The first contribution of this Thesis develops a certification argument for mixed-criticality certification upon multicores. The safety argument includes a set of safety mechanisms, diagnostic techniques and fault reaction features as required to comply with IEC 61508 functional safety standard upon a reference multicore architecture. The argument is built incrementally by transitioning from common-practice federated certification considerations to a mixed-criticality multicore solution.
- Secondly, as a proof of concept, we translate the generic assumptions and requirements of the safety argument into technical solutions by defining two alternative safety concepts for a railway (AAL+15) and an automotive (AAL+16) case study respectively. These safety concepts were reviewed and evaluated by an industrial certification authority. The former safety concept considers two multicore architectures, a complex COTS platform and an ad-hoc architecture. To deal with multicore challenges, the railway safety concept mainly relies on multicore partitioning and virtualization techniques through the use of a certified hypervisor. On the contrary, the automotive safety concept is based on the use of a set of hardware protection mechanisms that enforce separation in lieu of the hypervisor. This is achieved by a state-of-the-art automotive *certification-friendly* COTS multicore platform that embeds many valuable safety features in its architecture.

1.3.2 MBPTA Potential Adherence to Standards

MBPTA is an attractive alternative to existing timing analysis techniques to handle the complexity of multicore architectures. However, the innovation that MBPTA brings about is a major step from current certification procedures and standards. Currently, safety standards require quantifying the probability of failure of hardware components to assess the suitability of the design with respect to reference values for each integrity level. Software, on the contrary, is qualitatively assessed based on practical experience, by demonstrating coverage of all requirements of the standard.

- The third contribution of this Thesis is the quantification of the likelihood of *execution-time exceedance* events and relating it to target failure metrics in support of certification arguments, much like for hardware random faults. To this end, we evaluate the MBPTA approach with respect to IEC 61508 and we build on metrics such as failure rates and diagnostic coverage to determine the cut-off exceedance probability for the execution time of a task.

In spite of its potential benefits in complex systems, MBPTA cannot be naively used on every processor architecture as it requires certain timing properties that make the platform MBPTA-compliant. The use of MBPTA to analyze the timing behavior of safety-critical systems rests on its ability to derive trustworthy WCET bounds. This ability depends on the soundness of the MBPTA method per se, as well as on the satisfaction of safety requirements placed on the Pseudo-Random Number Generator (PRNG) that plays a key role in the platform-level randomization needed by MBPTA.

- We present the design of a low-area, low-power PRNG for multicore processors that meets IEC 61508 safety requirements for its use with MBPTA. PRNG's malfunction can compromise the safe operation and the availability of the system by invalidating the pWCET estimates derived for safety-related software. To avoid this, we adapt a reference PRNG design to a multicore architecture and define the safety mechanisms required to achieve IEC 61508 SIL 3 compliance ([AAH+15](#)).

1.3.3 Practical Multicore Certification Compliance

Certification standards reflect the state of practice in industry rather than the state of the art. As a result, they do not evolve as fast as technology and they do not provide explicit guidance for multicore architectures yet. It is a common pattern that certification guidance does not prescribe particular solutions but ensure the quality of any solution. Commonly, the user is expected to extrapolate

the guidelines and requirements in standards to the specifics of the particular processor under consideration. However, current certification guidelines are hard to extrapolate to multicore architectures that involve novel design paradigms like shared resources or high performance features not contemplated in the standards. On the road to addressing this limitation, recently, certification authorities in the airborne domain have published a position paper for multicore processors (CAST-32A) ([CAST-32A](#)). The CAST-32A position paper lists a set of objectives to help addressing multicore certification challenges and is, at the time of writing, the only explicit certification guidance for multicore architectures.

- The last contribution of this Thesis focuses on the analysis of the objectives of CAST-32A multicore certification guidance report, and discussion of the feasibility of achieving them with current COTS multicore design trends ([AAA+17](#)). We show potential limitations that embedded system designers may face in achieving those goals and tailor some of the generic principles in CAST-32A to a specific COTS multicore processor (the NXP's eight-core P4080 ([P4080](#))).

1.4 Thesis Organization

Each of the major contributions of this Thesis is presented in a different chapter, as visually depicted in Figure [1.1](#). Accordingly, this Thesis is structured as follows:

- Chapter [2](#) defines the basic concepts on safety certification, multicore architectures and timing analysis. In addition, it covers the necessary background by surveying mixed-criticality integration techniques and timing analysis solutions on multicores.
- Chapter [3](#) explains the experimental setup used for the evaluation of the contributions of this Thesis. In particular, it describes the multicore processors considered, the applications, and the timing analysis procedure and tools.
- Chapters [4](#) and [5](#) cover the certification of multicore mixed-criticality architectures. More concretely:
 - Chapter [4](#) defines the integrated multicore safety argumentation using the Goal Structuring Notation (GSN).
 - Chapter [5](#) applies and evaluates the safety argumentation on domain-specific safety concepts for railway and automotive use cases.

1.5 List of Publications

- Chapter 6 and Chapter 7 focus on the safety-critical industry readiness of MBPTA for the timing analysis of multicores by addressing its certification implications:
 - Chapter 6 advocates for the treatment of software timing faults as random hardware faults in standards.
 - Chapter 7 proposes a safety criticality-cognizant PRNG design.
- Chapter 8 evaluates the practical certification compliance of complex multicore processors with particular emphasis on CAST-32A certification guidance.
- Chapter 9 presents the main conclusions and potential future work of this Thesis.

1.5 List of Publications

This section collects the list of publications produced as a result of the research carried out in this Thesis:

1. **Fitting Software Execution-Time Exceedance into a Residual Random Fault in ISO-26262.** I. Agirre, F.J. Cazorla, J. Abella, C. Hernandez, E. Mezzetti, M. Azkarate-Askasua, and T. Vardanega. *IEEE Transactions on Reliability (to appear)*, 2018. DOI: [10.1109/TR.2018.2828222](https://doi.org/10.1109/TR.2018.2828222).
2. **On the Tailoring of CAST-32A Certification Guidance to Real COTS Multicore Architectures.** I. Agirre, J. Abella, M. Azkarate-Askasua, F.J. Cazorla. *SIES 2017 12th IEEE International Symposium on Industrial Embedded Systems*. Toulouse (France); June 14-16, 2017. DOI: [10.1109/SIES.2017.7993376](https://doi.org/10.1109/SIES.2017.7993376)
3. **Automotive safety concept definition for mixed-criticality integration on a COTS multicore.** I. Agirre, M. Azkarate-Askasua, A. Larrucea, J. Perez, T. Vardanega, F.J. Cazorla. *5th International Workshop on Next Generation of System Assurance Approaches for Safety-Critical Systems, SASSUR 2016 (35th SAFECOMP conference)*. Trondheim (Norway); September 21-23, 2016. DOI: [10.1007/978-3-319-45480-1_22](https://doi.org/10.1007/978-3-319-45480-1_22)
4. **A safety concept for a railway mixed-criticality embedded system based on multicore partitioning.** I. Agirre, M. Azkarate-Askasua, A. Larrucea, J. Perez, T. Vardanega, F.J. Cazorla. *13th IEEE International Conference on Dependable, Autonomic and Secure Computing, DASC 2015*.

Liverpool (United Kingdom); October 26-28, 2015. DOI: [10.1109/CIT/IUC-C/DASC/PICOM.2015.268](https://doi.org/10.1109/CIT/IUC-C/DASC/PICOM.2015.268)

5. **IEC 61508 SIL 3 compliant pseudo-random number generators for probabilistic timing analysis. I. Agirre**, M. Azkarate-Askasua, C. Hernandez, J. Abella, J. Perez, T. Vardanega, F.J. Cazorla. *18th Euromicro Conference on Digital System Design, DSD 2015*. Madeira (Portugal); August 26-28, 2015. DOI: [10.1109/DSD.2015.26](https://doi.org/10.1109/DSD.2015.26)

In addition, although they do not constitute explicit contributions of this Thesis, the following publications are tightly related to it.

6. **Software time reliability in the presence of cache memories. S. Milutinovic**, J. Abella, **I. Agirre**, M. Azkarate-Askasua, E. Mezzetti, T. Vardanega, F.J. Cazorla. *22nd International Conference on Reliable Software Technologies, Ada-Europe 2017*. Vienna (Austria); June 12-16, 2017. DOI: [10.1007/978-3-319-60588-3_15](https://doi.org/10.1007/978-3-319-60588-3_15)
7. **EPC enacted: Integration in an industrial toolbox and use against a railway application. E. Mezzetti**, M. Fernandez, A. Bardizbanyan, **I. Agirre**, J. Abella, T. Vardanega, F.J. Cazorla. *23rd IEEE Real-Time and Embedded Technology and Applications Symposium, RTAS 2017*. Pittsburgh (United States); April 18-20, 2017. DOI: [10.1109/RTAS.2017.19](https://doi.org/10.1109/RTAS.2017.19)
8. **Mitigating software-instrumentation cache effects in measurement-based timing analysis. E. Díaz**, J. Abella, E. Mezzetti, **I. Agirre**, M. Azkarate-Askasua, T. Vardanega, F.J. Cazorla. *16th International Workshop on Worst-Case Execution Time Analysis, WCET 2016*. Toulouse (France); July 5, 2016. DOI: [10.4230/OASIcs.WCET.2016.1](https://doi.org/10.4230/OASIcs.WCET.2016.1)
9. **PROXIMA: Improving Measurement-Based Timing Analysis through Randomisation and Probabilistic Analysis. F.J. Cazorla**, J. Abella, J. Andersson, T. Vardanega, F. Vatrinet, I. Bate, I. Broster, M. Azkarate-Askasua, F. Wartel, L. Cucu, F. Cros, G. Farrall, A. Gogonel, A. Gianarro, B. Triquet, C. Hernandez, C. Lo, C. Maxim, D. Morales, E. Quiñones, E. Mezzetti, L. Kosmidis, **I. Agirre**, M. Fernandez, M. Slijepcevic, P. Conmy, W. Talaboulma. *19th Euromicro Conference on Digital System Design, DSD 2016*. Limassol (Cyprus); August 31 - September 2, 2016. DOI: [10.1109/DSD.2016.22](https://doi.org/10.1109/DSD.2016.22)
10. **Temporal independence validation of an IEC 61508 compliant mixed-criticality system based on multicore partitioning. A. Larrucea**, **I.**

1.5 List of Publications

Agirre, C.F. Nicolas, J. Perez, M. Azkarate-Askasua, T. Trapman. 18th *Forum on Specification and Design Languages, FDL 2015*. Barcelona (Spain); September 14 - 16, 2015. DOI: [10.1109/FDL.2015.7306359](https://doi.org/10.1109/FDL.2015.7306359)

11. **A modular safety case for an IEC 61508 compliant generic hypervisor** A. Larrucea, J. Perez, **I. Agirre**, V. Brocal, R. Obermaisser. 18th *Euromicro Conference on Digital System Design, DSD 2015*. Madeira (Portugal); August 26-28, 2015. DOI: [10.1109/DSD.2015.27](https://doi.org/10.1109/DSD.2015.27)

Chapter 2

Background and Basic Concepts

This chapter provides the essential background and introduces the basic concepts on which the work on this Thesis is based. First, Section 2.1 defines the relevant concepts and terms. Then, subsequent sections review the most relevant literature: Section 2.2 summarizes existing solutions for mixed-criticality integration upon multicore architectures; next, in Section 2.3, we discuss multicore timing analysis solutions as a means to deal with the timing unpredictability and possible temporal interferences on multicore processors. On each section we analyze how the set of works consider safety certification. To conclude, Section 2.4 provides a summary of the state-of-the-art.

2.1 Basic Concepts

This section briefly introduces the main constituent elements of this Thesis, which span over three research fields: (i) safety-critical systems, (ii) multicore processor architectures, and (iii) timing analysis.

2.1.1 Safety-critical Embedded Systems

Among the CRTES domains, the main focus of this Thesis is on safety-critical systems. Various terms come into play when referring to safety-critical systems and their certification. However, some of them often have slightly different meanings or interpretations depending on the domain. Below we provide the definitions of the basic terms related to safety-critical systems and we clarify the taxonomy used throughout this Thesis.

Safety and Functional Safety

Safety is a dependability attribute – together with reliability, availability, security, survivability and maintainability – that can be defined as the absence of unacceptable risk leading to catastrophic consequences (ALR+04). In this work, we focus on the safety of electronic control systems, which is referred to as *functional safety* (IEC61508).

Certification and Standards

The deployment of safety-critical systems requires to comply with legal regulations in place on each country or state. This involves a *certification* process where an independent certification body must approve through a conformity assessment that the system is suitable and safe enough for its intended use (i.e., that it is compliant with the active legal directives). Currently, the most common certification approach is standard based, i.e., a conformity assessment is achieved by proving adherence to the applicable safety standards.

There is an extensive set of safety standards that provide guidance and recommendations to aid in the certification process. Despite most standards target domain-specific applications, many of them are based on the generic international IEC 61508 standard for the Functional Safety of Electrical/Electronic/Programmable Electronic (E/E/PE) safety-related systems, which is applicable across multiple industrial sectors. Some other domains, like avionics or space industries, establish their own set of independent standards (e.g., DO 178, European Cooperation for Space Standardization (ECSS)) (Figure 2.1).

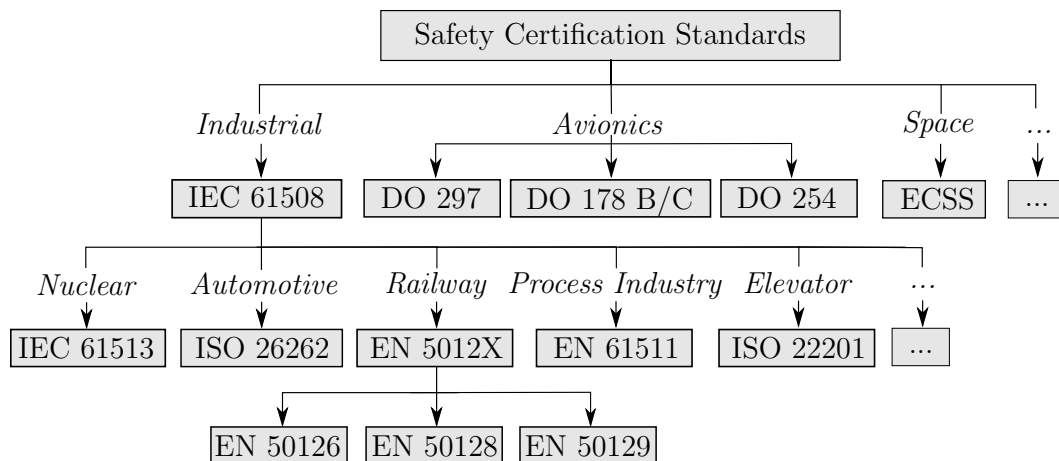


Figure 2.1: Relation among safety certification standards.

Safety standards define development processes with additional steps and re-

2.1 Basic Concepts

quirements to cover safety and certification needs. As an example, Figure 2.2 illustrates the overall *safety life-cycle* defined in the IEC 61508 standard (IEC61508). This life-cycle guides the developer into the adoption of special analyzes and safety management activities in all phases of the development process of the safety-critical system. The realization phase encompasses the system level safety life-cycle and hardware and software development processes. IEC 61508 recommends the V-model for designing safety-related software and hardware. The V-model requires a clearly structured design process: the outcome of each step is the input to the next step, until completing the implementation. This implementation is then verified against the corresponding design in each step (i.e., unit testing, integration testing and verification of the complete design).

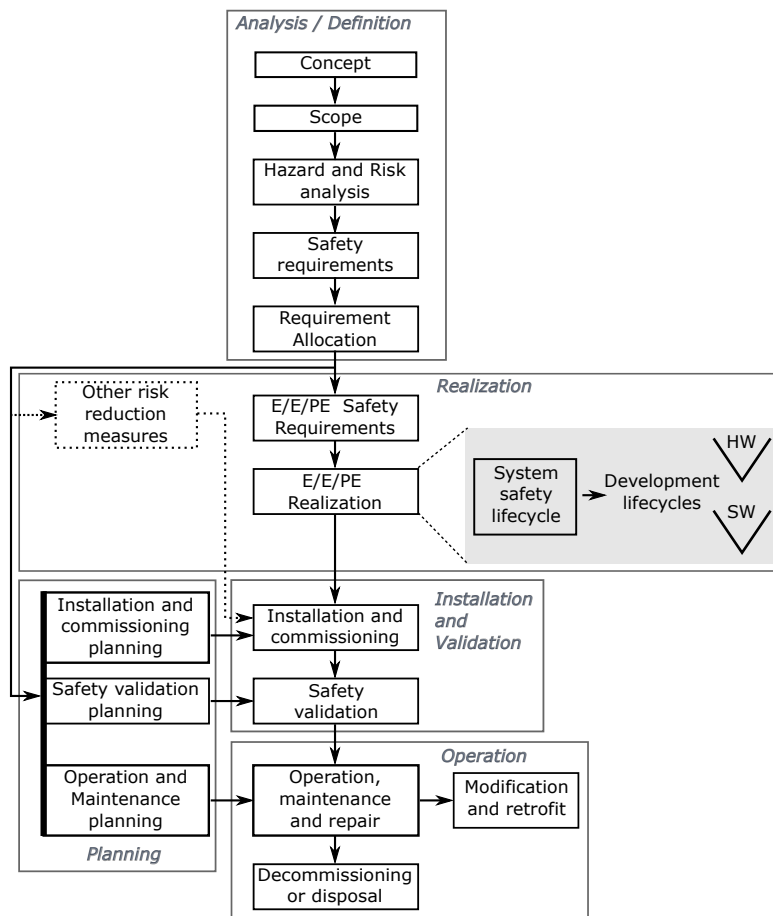


Figure 2.2: IEC 61508 safety life-cycle (IEC61508).

Criticality, Integrity or Assurance level

We use all three terms interchangeably to refer to the level of assurance and risk reduction needed against failures. The risk associated to the possible dangerous consequences (i.e., hazards) of a system determines the criticality level assigned to a given *safety function* that aims to reduce such risk up to tolerable rates. This criticality attribute is determined based on the severity, frequency of exposure and controllability of the hazardous events. The criticality level dictates the requirements imposed by standards in each of the phases of the safety life-cycle; with more restrictive measures for higher criticality levels. Different safety standards use distinct nomenclatures and levels, e.g., Safety Integrity Level (SIL), Automotive Safety Integrity Level (ASIL), Software Safety Integrity Level (SSIL), Design Assurance Level (DAL), etc. Similarly, even if they are out of the scope of this Thesis, security criticality levels are also defined (e.g., within Multiple Independent Levels of Security, MILS). Table 2.1 shows the criticality levels defined in different domains and an approximate equivalence among them.

Table 2.1: Criticality levels on different domain standards (Bou15).

Standard	Criticality level				
IEC 61508	-	SIL 1	SIL 2	SIL 3	SIL 4
ISO 26262	-	ASIL A	ASIL B	ASIL C	ASIL D
EN 50126/9	SIL 0	SIL 1	SIL 2	SIL 3	SIL 4
EN 50128	SSIL 0	SSIL 1	SSIL 2	SSIL 3	SSIL 4
DO 254/178	DAL E	DAL D	DAL C	DAL B	DAL A
ECSS-Q-ST-80C	-	D	C	B	A

For systems containing functionality with different safety implications and thereby, different assurance levels, we use the term *mixed-criticality*.

Fault, Error and Failure

A *fault* is a (hypothesized) cause (abnormal condition or defect in an item) of an error. Subsequently, an *error* is defined as an incorrect state of an entity that can, therefore, cause a failure. A *failure* is a deviation from the correct service of the entity given by the infringement of the specifications (ALR+04). The definitions of these three terms are, however, dependent on system boundaries. The entity may be either a single component, a subsystem or a system and a failure of a component can be considered as a fault at a higher level of the system hierarchy. A *Fault Containment Region (FCR)* defines the boundaries of an immediate failure impact, i.e., it comprises the set of components that fail independently from each

2.1 Basic Concepts

other. FCR are commonly different for hardware (*physical FCR*) and software (*design FCR*) faults.

Whenever failures are deterministically related to a certain cause (e.g. a design fault either in hardware or software), standards refer to them as *systematic*. On the contrary, when a failure is the consequence of an arbitrary event (e.g., hardware degradation) standards classify it as *random* (IEC61508; ISO26262; EN50126). The probability of failure of the latter is quantified by the *failure rate*.

Considering the case of timing as an illustrative example, a design (timing) fault can, for instance, be a bad (optimistic) WCET estimation (i.e., the estimated WCET is below the real WCET). This fault may get activated in the form of a deadline miss and lead to an error state that can cause a system failure. For instance, the example depicted in Figure 2.3 shows that, due to a design timing fault, a task may need more time to execute than that allocated to it in the scheduling. In the event of a deadline miss, this causes a context switch to happen before the task completes its execution (e.g., before it writes the outcome of the computations into the output ports). As a consequence, the outputs are not updated in time and may contain obsolete data until next period. Within this time interval, the system is in an erroneous state. Finally, this error may propagate into an improper commanding of the actuators and, thereby, cause a systematic functional failure (e.g., unexpected airbag activation in a car) or a systematic timing failure (e.g., late activation of airbag system).

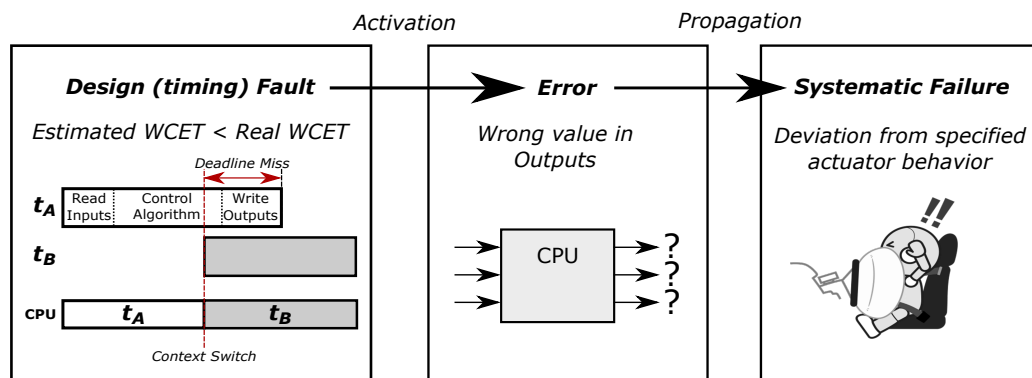


Figure 2.3: Illustrative example of (timing) Fault, Error and Failure.

Fail safe and fail operational systems

The term *fail safe* is used for those safety-critical systems that, in case of a system failure, they can transition to a *safe state* either by the safety function or diagnosis. A system is said to be in the *safe state* when it reaches a state free from any possible hazardous condition. For example, the safe state of a railroad barrier system is

setting it in the stop position. On the contrary, if a system does not dispose of a safe state during mission time, we refer to it as *fail operational*. For instance, in the avionics domain, it may not be possible to define a safe state for some systems, such as, a flight control system that must provide some minimal level of service even in the presence of faults ([Kop11](#)).

2.1.2 Multicore Processors

Multicore processors comprise two or more processing cores implemented in the same integrated circuit or silicon die. Tasks can run simultaneously on the different cores of a multicore. While there are a plethora of different multicore implementations, most of them are built on the same pattern. The cores in a multicore typically share platform level elements, such as, main memory, peripheral devices and the interconnect to communicate with those components. Figure 2.4 sketches the main components of a generic COTS-based multicore processor:

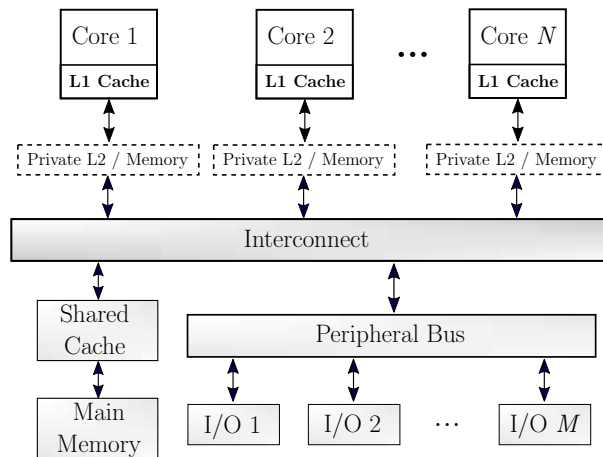


Figure 2.4: Generic COTS based multicore architecture.

Private Resources

Private or core-local resources in a multicore are those resources attached to, or for the exclusive use of, each of the cores in the processor (i.e., the resources that, by design, can not be accessed by more than one core). Private resources include the *processing cores* and *first level caches*. Depending on the particular implementation, some architectures also include additional private memory, like the P4080 ([P4080](#)) that includes a private second level cache or the AURIX ([AURIX](#)) that has private scratchpad memories attached to each core.

2.1 Basic Concepts

If all processing cores in the multicore share the same architecture, the multicore is regarded as *homogeneous*. Examples include NXP's QorIQ family with power processor cores like the P4080 (P4080), ARM based layerscape (NXPLS) family, and most of Intel's solutions (e.g. Xeon (XEON)). On the contrary, a multicore is *heterogeneous* if its cores are of two or more different processor architectures. NXP's I.Mx family (NXiMX) and Texas Instrument's Concerto microcontroller family (TICon) are examples of heterogeneous multicore architectures.

In the recent years, the safety-critical domain has started using *lockstep multicore* processors. A lockstep processor is a special type of multicore – typically homogeneous dual-core – which is specifically designed to achieve high reliability. In such architecture, both cores execute the same program with the same inputs. A hardware comparator is continuously comparing the outcomes of the computations among the two cores to detect any discrepancy and hence, possible hardware faults. Accordingly, in this set-up, the additional core does not provide increased performance but improved reliability.

Shared Resources

Most COTS multicore architectures are shared-memory based. For instance, they typically include a second (or higher level) *shared cache*, also referred as *platform cache*, and a shared *main memory* or *platform memory* connected to the cores through an *interconnect*. Depending on the particular implementation, the interconnect can be implemented as a traditional *system bus* or *crossbar* or by means of a *Network-on-Chip (NoC)*. The communication between the platform memory and the cores is handled by one or more *memory controllers* and an *arbiter*, generally attached to the interconnect. The interconnect is hence the interface between the cores and rest of platform components and all coherency traffic, memory transactions, peripheral accesses and interrupts flow through it. In addition, there is a wide range of possible *I/O peripherals* that a multicore can include depending on the particular design and application domain. Peripherals are usually connected to the interconnect through a *peripheral bus*.

It shall be noted that a multicore is different from a *multiprocessor* architecture, although the latter also integrates multiple processing elements, they are not always attached to the same silicon die and they commonly integrate a distributed memory system. Similarly, when the number of cores is much larger (in the order of hundreds), referred to as *manycore*, they are typically interconnected through NoC rather than bus interconnects.

Software distribution

Based on the software distribution, multicore systems are categorized as *Symmetric* or *Asymmetric* Multiprocessing:

- **Symmetric Multiprocessing (SMP):** In SMP a single Operating System (OS) is distributed across all cores in the system. The OS manages the programs executing on top of it and assigns them to the cores. Some operating systems further allow to restrict which applications are allowed to run on each specific core.
- **Asymmetric Multiprocessing (AMP):** In AMP each core may or may not independently execute a different OS.

2.1.3 Timing Analysis

CRTES typically pose severe timing restrictions on the execution time of their functions usually derived from the requirements of the control algorithms responsible of commanding the system (e.g., the control of a braking system in a vehicle). Real-time control systems must react to stimuli from the environment and produce results before they reach a time instant called *deadline* (for example, a steering system shall react within 50 *ms* (Kop11)). To guarantee the satisfaction of such constraint, it is required to establish upper-bounds on the execution times of real-time tasks (i.e., WCET estimates) during software development. WCET estimation is a basic precondition for schedulability analysis and hence, for guaranteeing that all tasks will meet their deadlines in the designed real-time system. The computation of these timing parameters is part of the process referred to as timing analysis (WEE+08; Kop11).

Worst-Case Execution Time (WCET)

The worst-case execution time (WCET) of a task depends on many factors such as the source code, programming language, compiler, the input data and underlying hardware architecture. In fact, as the hardware has many state-full resources, the same binary with the same inputs may have different execution times in different consecutive runs. For instance, a cache memory takes advantage of the execution history to obtain performance gains from run to run. As a result, the execution time of a task is not a fixed value but a complex distribution as shown in Figure 2.5.

At the time of analyzing the worst-case timing of a system, in most cases, it is not feasible to predict all possible conditions leading to worst-case execution time due to the extremely large state space (WEE+08). As depicted in Figure 2.5,

2.1 Basic Concepts

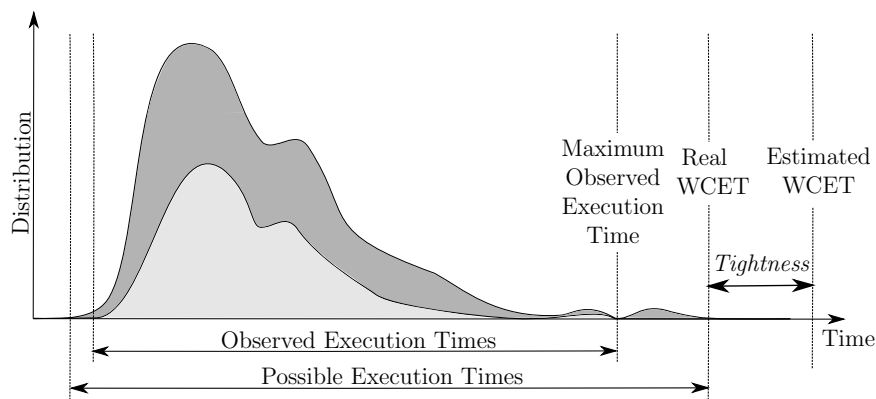


Figure 2.5: Basic concepts for WCET analysis (WEE+08).

commonly three different bounds are defined (equivalent bounds apply for the Best-Case Execution Time):

- The *Maximum Observed Execution Time (MOET)*, or High-Water Mark (HWM), refers to the largest execution time value obtained by exhaustively exploring the system, usually by collecting end-to-end measurements of a task under different inputs and system states.
- The *Real WCET* is, by definition, the maximum time interval between task activation and task termination that can occur during system operation (i.e., the maximum of all possible execution times for a task).
- The *Estimated WCET* is the result of the timing analysis to upper-bound the real WCET.

As illustrated in Figure 2.3, a task not finishing its execution before its estimated WCET expires may result in a *temporal failure*. As a result, it is crucial that the estimated WCET value upper-bounds the application’s real WCET and that it is, therefore, a *trustworthy* upper-bound. In addition to trustworthiness, *tightness* is another vital property of timing analysis (AHP+14). The tightness of an execution time bound refers to the deviation of the analysis results (i.e., estimated WCET) from the real WCET. The bigger the difference, the larger the overestimation of the analysis method. The tightness of an estimation is usually dependent on the *timing predictability* of the system, that is, on the hardware architecture, complexity of the software and the method used for timing analysis.

Classification of Timing Analysis Techniques

Several methods have been adopted to compute such estimates on single-core processors, which can generally be classified as *static (STA)*, *measurement-based*

(*MBTA*) or *hybrid (HYTA)* according to the procedure followed to obtain WCET estimates:

- **Static Timing Analysis, STA:** Static methods analyze the source code together with some (abstract) model of the hardware architecture and compute WCET upper-bounds without executing the task or set of tasks under analysis on the target hardware platform.
- **Measurement-based Timing Analysis, MBTA:** Measurement-based methods rely on end-to-end execution time observations obtained by executing the program (or parts of it) on the real hardware or a simulator under different execution conditions (e.g. different test vectors).
- **Hybrid Timing Analysis, HYTA:** A combination of STA and MBTA.

In addition, each method can also be grouped as *Deterministic Timing Analysis (DTA)* or *Probabilistic Timing Analysis (PTA)* depending on the overall approach and the nature of the computed results ([AHP+14](#)). DTA and PTA techniques can either be static, measurement-based or hybrid, so in summary, according to the taxonomy defined in ([AHQ+15](#)), there are six groups of timing analysis techniques (summarized in Figure 2.6).

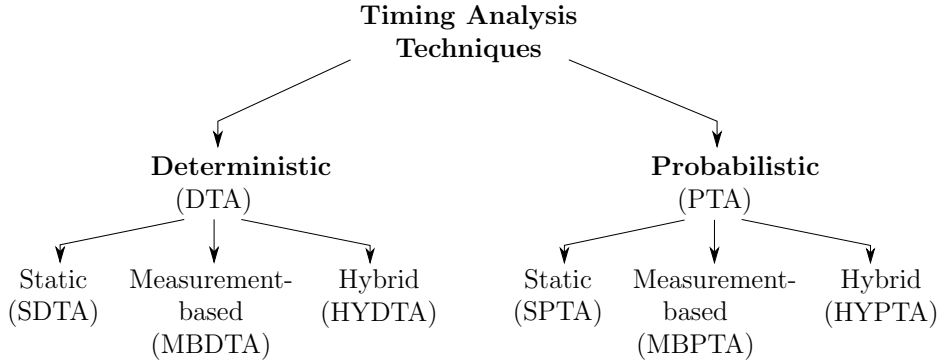


Figure 2.6: Classification of timing analysis techniques ([AHQ+15](#)).

- **Deterministic Timing Analysis:** DTA techniques calculate a single WCET estimate (as the estimated WCET bound depicted in Figure 2.5).
- **Probabilistic Timing Analysis:** PTA techniques obtain a probability distribution for the WCET, called probabilistic WCET (pWCET). The pWCET distribution function determines a WCET value (e.g. $7ms$) associated to a cut-off probability of exceedance (e.g. 10^{-9}).

As mentioned before in this document, the focus of this Thesis is on the MBPTA approach as it is closer to industrial practice than its static counterpart.

Measurement-Based Probabilistic Timing Analysis

PTA techniques generate a pWCET distribution function that upper-bounds the execution time of the program under analysis. Figure 2.7 shows an example of the pWCET outcome of PTA for a given task. The pWCET function allows to determine the WCET upper-bound (e.g., 7.8 ms in Figure 2.7) for a given cut-off probability of exceedance (e.g. 10^{-14}), which can be arbitrarily low to fit with the requirements of the application domain.

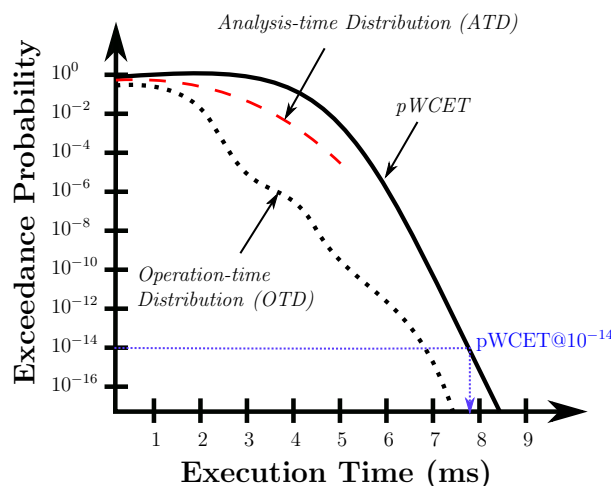


Figure 2.7: MBPTA concepts and example pWCET outcome.

Regarding to its measurement-based variant, i.e., MBPTA, the pWCET projection is computed by collecting a number of end-to-end execution time measurements on the target hardware – Analysis-time Distribution (ATD) in Figure 2.7 – and applying probabilistic analysis methods – i.e., Extreme Value Theory (EVT) (CSH+12) – to them to determine the probability of extreme values to occur (the tail of the pWCET distribution in Figure 2.7). PTA techniques require that the events under analysis, execution time observations in this case, have a distinct probability of occurrence and are independent and identically distributed (i.i.d.)¹ (CAA+16; KQA+16). In addition, EVT predicts the extreme timing behavior of just the set of observations passed to it, and hence, the resulting pWCET estimation is only valid for the execution conditions represented by those observations. As a consequence, the quality of WCET estimates depends on the *representativeness* of measurements, i.e., MBPTA requires that the set of measurements

¹Independence among two random variables is attained when the occurrence of one event does not impact on the occurrence of the other event. Two random variables are said to be identically distributed if they have the same probability distribution (KQA+14).

collected in the target platform at analysis time is representative of all possible conditions that may arise during operation (Operation-time Distribution (OTD)). The representativeness issue applies to both hardware (i.e., jittery platform resources) and software (e.g., program execution path). Unfortunately, conventional processor designs fail to provide these properties (KQA+14; KQA+16).

MBPTA has been shown to be applicable by adapting the platform to introduce the concept of *time-upper bounding* and *time randomization* in the hardware/software architecture (KQA+14; KQA+16; KVM+16; KCQ+13).

- **Time Upper-bounding:** This technique forces selected jittery hardware resources to work at their highest latency during analysis. Time upper-bounding is implemented in those hardware resources whose timing behavior depends on elements beyond the reach of hardware (COBHAM; HAC+17). The floating point unit provides an illustrative example: the latency of its operations in fact depends on the operands; that is, multiplying a value by 0.0 can take shorter than with other parameters. MBPTA’s prescription to force the floating point unit to work at its highest latency (per operation type) during analysis ensures that operation conditions cannot lead to higher execution times than those observed at analysis and hence, a single run suffices to capture their worst-case operation-time behavior. Time upper-bounding is also considered for other resources such as, for instance, the number of contenders considered for arbitration in a shared bus.
- **Time Randomization:** This technique causes the response time of some jittery resources to exhibit a probabilistic behavior that also holds during operation. It is implemented in processor resources whose timing behavior depends on the structural dependencies created by the hardware design (KQA+14; HAC+17). Randomization helps remove those dependencies, which have no bearing on the program semantics. For instance, whether two addresses compete for the same cache space depends on how they are mapped to cache lines. Cache mapping can be randomized to make conflicts to occur probabilistically (KAQ+13b; KQA+16). Time randomization can also be applied to the bus arbiter, to randomly choose the core that is granted access to a shared resource (JKA+14). As a result, a representative distribution of the impact that jittery resources may cause on execution time can emerge after a statistically-significant number of observation runs.

In addition, solutions also exist to deal with the jitter caused by the software program flow (i.e., execution paths). The Path Upper-Bounding (PUB) technique (KAW+14) and the Extended Path Coverage (EPC) (ZMV+15) technique that builds upon the former, upper-bound the execution time of any path of the program by observing end-to-end traversals of it in MBPTA-compliant platforms. In

this way, both i.i.d. and representativeness conditions are fulfilled. Figure 2.7 illustrates this notion. The dotted line depicts the Empirical Complementary Cumulative Distribution Function (ECCDF) of the OTD¹, and the dashed line the ECCDF of the ATD. MBPTA design modifications guarantee that the ATD upper-bounds the OTD by construction.

2.2 Mixed-criticality Integration on Multicores

Besides the certification process does not change with the advent of multicore processors, existing multicore architectures bring significant limitations for achieving certification. In fact, current standards provide very limited guidance on how to tackle the challenges of modern multicores when used in CRTES. On the contrary, standards do consider the co-existence of mixed-criticality functions and provide two options for their certification: (i) certify all components of the system – including the non critical software – for the highest criticality level present on the system, or (ii) provide enough evidence of independence among the applications of different criticality levels and allow, in this way, the certification of each application according to its criticality level. The cost and effort of certifying a system increases considerably with the criticality level (SEH+12). Consequently, the cost and certification effort of the former approach is usually prohibitive. As a consequence, it is crucial to prevent lower criticality applications from adversely affecting higher criticality ones to guarantee such independence.

The challenge of achieving such independence of execution in mixed-criticality systems is highly dependent on the implementation architectural paradigm, that can either be *federated* or *integrated*:

- **Federated Mixed-criticality system:** Critical functions of a given criticality level are allocated to dedicated hardware, physically separated from the rest of the system with different criticality implications. This “one function = one device” paradigm provides physical separation by design.
- **Integrated Mixed-criticality system:** Critical functions of a given integrity level co-exist – in the same embedded platform – with subsystem of different assurance levels or even with no critical implications. In this implementation, the separation requirement involves considering additional mechanisms due to new challenges, such as, the management of the use of shared resources among the applications.

¹We plot OTD for illustrative purposes only, since its retrospective nature (which observes operation-time events) is too late to feed WCET analysis.

Some application domains have adopted standardized integrated architectures, such as, IMA ([DO297](#)) in avionics or AUTOSAR ([AUTOSAR](#)) in automotive and there are ongoing initiatives in other domains such as, Safe4Rail project in Railway ([S4R](#)).

Standards provide some guidance for achieving independence among the different subsystems allocated to the same chip based on the concept of partitioning. Partitioning, firstly adopted in the ARINC-653 standard ([ARINC653](#)), is a common approach for the implementation of integrated mixed-criticality systems. Partitions provide functional separation of the applications and design fault containment ([Rus99](#)). To that end, a partition encapsulates the physical resources both spatially and temporally:

- **Spatial independence** must ensure that one application does not alter the code or private data of another application, nor command the private devices allocated to them. A common approach proposed in standards is using memory protection mechanisms like Memory Protection Unit (MPU) or Memory Management Unit (MMU).
- **Temporal Independence** shall ensure that the execution of one application does not obstruct, in the temporal domain (including performance, latency and jitter), the execution of other applications. For achieving temporal independence standards suggest using fixed and cyclic scheduling of partitions (IEC 61508-3 Annex F ([IEC61508](#)), ISO 26262-6 Annex-D ([ISO26262](#)), ARINC-653 ([ARINC653](#)), DO 178 6.3.3f ([DO178](#)), CAST-32A ([CAST-32A](#))).

The most popular approach for partitioning and providing functional separation among mixed-criticality applications relies on virtual partitions running on top of *hypervisors*. Hypervisors, also known as Virtual Machine Monitors (VMM), can be defined as layers of software that establish independent execution environments (i.e., partitions) in a single computer platform. They can be classified as Type 1 bare-metal hypervisors, which lay directly on the bare-metal hardware or Type 2 hypervisors that run on top of an OS. Additionally, full-virtualization and para-virtualization hypervisors are distinguished. The former provides a complete virtual image of all hardware resources and thus, the software above it runs without modifications as it is unaware of the virtualization. The latter, introduces the concept of hypercalls or hypervisor services to request privileged operations. This requires customization of the guest program or OS in order to support those hypercalls.

2.2.1 Partitioning on Multicore Processors

One can implement partitioning strategies either by software or hardware mechanisms. The former is achieved by introducing additional abstraction layers (e.g., separation microkernel, hypervisor) into the system. For the latter, specific hardware designs attain partitioning and independence by avoiding interferences, or by special hardware extensions that protect the execution time and memory space of each application.

Software Techniques

The design of most hypervisors on the market today emerged from scratch to support the fields of servers and workstations. This means that they contain a large and complex code base not intended to be certified. For the embedded market, hypervisors compete to offer low footprint and overhead as well as efficiency, good isolation and real-time capabilities. Most of the hypervisors that particularly target embedded applications use bare-metal para-virtualization given the reduced overhead of this technique. Additionally, if embedded applications are part of critical systems, the deployment of the hypervisor involves certifying it according to the highest integrity level present on the system and hence, the simplicity of the virtualization layer is pursued.

In this line, the design of many embedded hypervisors specifically addresses safety certification on single-core architectures as listed in the survey provided by the authors in (ZQ12). However, these solutions are not easily scalable to modern multicore processors. Several hypervisor providers and research projects seek to adapt existing embedded hypervisors, such as, the PikeOS hypervisor (PikeOS), the open source XtratuM hypervisor (XtratuM) or the Wind River Hypervisor (WINDRIVER) to multicores. PikeOS is the first hypervisor achieving multicore certification for a dual-core architecture according to EN 50128 (PikSIL4; Fis14), however, it does not yet consider the parallel execution of critical applications. XtratuM is designed for embedded real-time systems and provides a set of properties to support certification (MRC11). It has been evaluated in a specific ad-hoc multicore system as part of the MultiPARTES project (CCM+14). Wind River provides a Safety Profile on top of VxWorks (VxSP15; VxCer15) which TÜV SÜD has positively assessed up to SIL 3 (IEC 61508) on single-cores. For the avionics market, Wind River has developed the ARINC 653-compliant VxWorks 653, suitable for multicore architectures on its 3.0 version (Vx653). The authors in (RM14) focus on meeting the isolation requirements for the integration of a mixed-criticality application on the Infineon AURIX multicore platform using a microkernel based hypervisor developed by ETAS Ltd. (RTA-HV).

All hypervisor-based solutions have many common features, such as, a para-

virtualization layer with higher privilege rights, MMU/MPU management for enforcing spatial separation on memories, statically defined cyclic scheduling patterns that assign available resources to tasks based on time partitions (based on ARINC specification), inter-partition communication mechanisms and health monitoring features for diagnosis. However, parallelization and resource sharing is often compromised to ensure safety, for instance by assigning exclusive time partitions for critical tasks (single-core like execution) (Fis14). Overall, safety-related hypervisors rely on cyclic ARINC based scheduling for dealing with temporal interferences but they do not provide the details on how the execution time bounds of that scheduling regime are computed. This is a big gap, since the temporal independence at application level is contingent on the timing behavior exhibited by the underlying multicore processor.

Hardware Techniques

As an alternative to software partitioning, hardware partitioning is enforced by predictable ad-hoc hardware architectural approaches specifically designed to attain separation and predictability. Approaches such as the ACROSS MPSoC (SEH+12), also referred as Time-Triggered System-on-Chip (TTSoC), and the CoMPSoC Multi-Processor System-on-a-Chip (MPSoC) (HGB+09; GAC+13) provide dedicated hardware resources for partitions that are communicated by means of a deterministic NoC rather than the buses present on modern multicore architectures. Thanks to the network infrastructure, a message-based communication centric approach is built, which relies on statically dimensioned Time-Division Multiplexing (TDM) algorithms to improve temporal determinism. However, the authors of these hardware-based separation platforms build their approaches based on multi-processor designs rather than multicores.

The major drawback of these approaches is that they require novel specific architectures. As a result, the implementation technology of these hardware approaches is often hard to deploy in industry. FPGAs require an extra certification effort and ASICs are an expensive technology. A promising COTS multicore is considered by the authors in (FSD+13), where they apply hardware modifications to the state-of-the art automotive Infineon TriCore AURIX platform (AURIX) to improve isolation.

Mixed Techniques

To reduce the overhead put on the hypervisor, today several COTS processors provide Instruction Set Architectures (ISA) with support for virtualization to use it in combination with hypervisors. These ISAs support virtualization by features such as, an additional CPU privilege mode for the hypervisor, memory protection

2.2 Mixed-criticality Integration on Multicores

mechanisms and I/O protection. For instance, Wind River hypervisor solutions (VxWorks) take advantage of the Intel virtualization support (Intel-VT) ([Int09](#)) and NXP provides virtualization features in both hardware (PowerPC with hardware assists for virtualization) and software layers (Freescale hypervisor) to use them in combination ([Free09](#); [Free08](#); [Bos13](#)).

A novel mixed hardware-software approach is the MultiPARTES platform. It comprises a heterogeneous multicore architecture (Intel dual-core ATOM (x86) and a muticore LEON 3 processor prototyped on an FPGA) and the XtratuM hypervisor in asymmetric multiprocessing (AMP) configuration ([SAA+14](#)). A Time-Triggered Network-on-Chip (TTNoC) implemented on the FPGA logically interconnects the cores taking the TTSoC architecture as a reference. At the software layer, the XtratuM hypervisor manages spatial and temporal independence. However, in the current approach, multicore contention in shared resources could produce interferences in the execution time duration of partitions. In order to achieve certification, the designer shall bound this interference and consider it in the cyclic scheduling plan design ([PGN+14](#)).

2.2.2 Certification

The difficulties to certify safety-critical and in particular mixed-criticality systems highly depend on their architectural properties, and therefore on systems' complexity. Current multicore processors fall into a "highly complex" category according to the European Aviation Safety Agency (EASA) ([EASA12](#)), as they include more than one Central Processing Unit (CPU) that depend on each other. In 2014, the certification authorities of the Federal Aviation Administration (FAA) presented their conservative position with respect to the certification of multicore processors in a position paper (CAST-32) ([CAST-32](#)). In particular, they consider multicores with only two active cores. The major concerns of the authors concentrate on the non-determinism caused by a number of features not present in single-core processors (e.g., interference channels caused by shared resources), a greater architectural complexity and undocumented and untestable features that may lead to an undesired behavior. Recently, the FAA/EASA have published an update of this paper, CAST-32A ([CAST-32A](#)), extending its applicability by removing the "only two active cores" restriction. CAST-32A provides certification guidelines based on approaches accepted by certification authorities in projects using multicore technology in Airbone systems.

A vast amount of research works target to enable the use of multicore architectures on mixed-criticality applications ([Kin09](#); [Huy12](#); [NP12](#); [BK14](#); [GK15](#)). The work described in ([Kin09](#)) lists per-component certification challenges (in shared caches, memory controllers, interconnect, etc.). In ([BK14](#)) they describe the multicore extensions of AUTOSAR. Similarly, the authors in ([Huy12](#)) discuss the impact

of multicore architectures in ARINC-653 based systems. However, multicore contention is an uncovered issue that needs to be resolved. In the research conducted by Geiger et al. (GK15), the unpredictability and difficulty to accurately estimate WCET in modern multicores results in no current multicore suitable enough for achieving the required level of independence for mixed-criticality and for this reason they propose the use of an external monitoring device.

2.3 Timing Analysis on Multicores

As exposed in the previous section, most partitioning solutions rely on cyclic scheduling algorithms for achieving temporal independence. Being WCET bounds a basic pre-condition for designing the scheduling, determining upper-bounds on execution times of tasks is of utmost importance for ensuring temporal independence on multicores. In addition, in a certification process, the applicant shall provide evidence of the *timing correctness* of the systems, i.e., guarantees that tasks will finish within their corresponding deadlines. However, deriving WCET estimates is a known challenge, specially when it comes to modern multicore architectures (AHQ+15; KP08; MV11). There is extensive recent literature that seeks to address multicore WCET estimation, a detailed summary of existing works can be found in (PDK+15; FAQ+14).

2.3.1 Minimizing Multicore Contention

On the one hand, several authors propose hardware designs to limit multicore interferences and favor time predictability (PQC+09; SEH+12). A common approach rests on time-multiplexing shared resource usage with Round-Robin (PQC+09) or Time-Division Multiple Access (TDMA) (RAE+07; PAH+15) arbitration of requests. Cache partitioning is another common solution given the difficulties that shared caches entail in timing analysis. Several works (CWK+15; LPS12; KVC+14) study the impact of cache partitioning in mixed-criticality applications where they try to balance the conflicting requirements of partitioning for safety and sharing for increased efficiency through criticality-aware allocation mechanisms.

On the other hand, as COTS multicore architectures are not generally time-predictable, researchers are developing software solutions to limit multicore contention. For instance, the authors in (NPB+14; NPH+14) suggest to limit the number of permitted accesses that each partition issues in a given time interval and use single-core like timing analysis techniques to compute an interference sensitive WCET. Similarly, other works prevent read/write phases of tasks to simultaneously use shared resources (e.g. interconnect) to prevent contention (BDN+16).

2.3.2 Accounting for Multicore Contention

Approaches relying on static analysis to derive contention bounds entail many limitations in multicore COTS architectures due to the complexity of modeling high-performance features and the lack of hardware details to do so. As an illustrative example, the authors in (NPB+14) base their approach in static timing analysis techniques, but the evaluation of their approach on a real processor (the P4080) exposes the need for measurements to quantify the access latencies in which their interference model is based on (NP12).

In the case of measurement-based techniques, which are currently the preferred industry choice, the challenge rests on guaranteeing that the worst-case is encountered during measurements and that the obtained value is therefore sound. Often, measurement-based techniques use resource stressing benchmarks (FGQ+12; NP12; BGGM14; BGG+14). The duty of benchmarks is generally to expose the timing behavior of certain resources of an architecture. Authors in (BGGM14; BGG+14) complement stressing benchmark analysis with an application specific evaluation and timing analysis. Similarly, the work reported in (FJA+15) increases the confidence of application measurements by complementing them with resource stressing benchmarks. Typically, industry addresses uncertainty in MBTA techniques by adding a safety-margin to the estimated execution time bound (e.g., commonly 20% margin). However, determining the size of this margin (which is qualitative rather than quantitative) and providing a formal proof for its justification is very difficult, even more in multicore architectures where the margin should cover all uncertainties.

In this line, probabilistic approaches allow to select a WCET bound based on its exceedance probability and thereby provide a formal justification for the chosen margins. Research works on MBPTA (KAQ+13a; CSH+12) show the modifications introduced in the hardware/software architecture of processors to apply MBPTA techniques and hence, make them *MBPTA-compliant* (KQA+14). Time randomization and upper-bounding techniques are implemented either in hardware (KAQ+13a) or software (KVM+16; KCQ+13). The outcomes of the PROAR-TIS (CQV+13; WKL+13; KQAF+14) project demonstrate the successful application of PTA upon single-core architectures and in PROXIMA (PROXIMA) its measurement-based variant, MBPTA, was ported to multicore processors (KQA+16; CAA+16).

2.3.3 Certification

Standards do not generally impose specific requirements on the timing analysis method itself. In fact, there are few allusions to timing analysis in standards. For instance, taking IEC 61508 safety life-cycle of Figure 2.2 as reference, the

standard only mentions timing analysis in specific verification requirements of the hardware and software development processes. In spite of these allusions to timing analysis, there is no guidance on which timing analysis technique is the most appropriate neither on how the developer should compute estimates nor on how one should verify the quality of the resulting WCET (GB13). Some timing analysis tools, such as, AbsInt’s abstract interpretation (aiT) (AbsInt; KF12; KPG+14) or Rapita’s RapiTime timing analysis tool (RPT; RPT178), have already been evaluated and qualified against safety certification standards on their single-core solutions. However, the suitability of the aiT static timing analysis tool is subject to the predictability of the platform and hence, may not scale well to multicore architectures. A model argument that explains the suitability of MBPTA for single-cores was constructed by the authors in (ZAV13). In general, the multicore approaches for WCET estimation, such as MBPTA, are still in their infancy and due to their novelty, there are no certification evidences yet.

2.4 Summary and Conclusions

In this section we have surveyed the extensive existing literature concerned with the hindrances that the transition to integrated multicore architectures involve for mixed-criticality systems. However, there are no established approaches yet to achieve their safety certification. Safety standards do not provide sufficient guidance yet for multicore implementations. When integrating mixed-criticality systems, safety standards require to guarantee temporal and spatial independence. Nonetheless, as reviewed in Section 2.2, the techniques adopted for single-core integration do not scale well to modern multicores.

Among the analyzed approaches, time deterministic processors and hardware based separation would ease the certification process by providing predictable timing guarantees and separation by design. In turn, they require completely new hardware designs (which are generally based on multi-processor architectural paradigms with distributed memory systems as opposed to modern COTS multicore designs) that often render difficult their deployment. On the contrary, hypervisors are an attractive solution from an industrial perspective to implement them on top of COTS multicores. However, hypervisors imply an additional software layer that introduces some overhead, compromises the simplicity of the overall system and it is required to be certified according to the highest integrity level present on the system. Overall, most hypervisors rely on ARINC-based cyclic scheduling regimes. However, ARINC specification does not yet define how such scheduling scheme should be translated to multicores where various tasks run in parallel on different cores. As a result, there are no established approaches yet to achieve mixed-criticality certification on multicore architectures.

2.4 Summary and Conclusions

Moreover, most solutions assume that tasks are derived WCET bounds before carrying out the scheduling. In multicores, the WCET of a task may considerably increase when it is running in parallel with other tasks though (NP12). Thereby, this approach only holds when all resources are partitioned in such a way that the load that co-runner applications put on shared resources does not increase tasks' WCET (i.e., WCET estimates are fully time-composable). Under these circumstances, the correctness of the hypervisor based solutions implicitly depends on the correctness of the WCET estimates.

As a consequence, deep research activities are being held for dealing with multicore contention and WCET estimation as summarized in Section 2.3. Some approaches prevent the simultaneous use of shared resources to improve predictability and independence at the cost of compromising performance. This requires to predict and control the maximum interference among co-running applications. The common design principle of adding a conservative safety-margin to estimates becomes unrealistic for multicores, due to their increased timing uncertainties (w.r.t. single-cores) and the subsequent difficulties at providing a formal argument for the chosen margins. Probabilistic analysis methods are an emerging alternative for replacing the 'qualitative' safety-margin by formal quantitative justification. Nevertheless, MBPTA requires to adopt *MBPTA-friendly* processor design principles.

In conclusion, unless all inter-core interferences in a multicore are minimized (which is unlikely to be adopted by hardware manufacturers and difficult to realize in software), achieving temporal independence by other solutions, like hypervisors, requires a detailed analysis of the timing behavior exhibited by the underlying multicore processor. Considering the individual drawbacks of the different timing analysis approaches, it is currently not possible to establish a general advice on which timing analysis technique to use for multicore mixed-criticality system certification, as it is still an open technical challenge.

Chapter 3

Experimentalt Setup

This chapter presents the main components involved in the evaluation framework of this Thesis: (1) the multicore processors considered, (2) the applications and (3) the timing analysis procedure and tools. These components and their dependencies with this Thesis' contributions are illustrated in Figure 3.1 where the elements developed in the scope of this Thesis are highlighted in green color. First, the contributions related to safety are theoretically evaluated by the formulation of safety concepts on industrial case studies for their formal review by a certification body. Then, execution time measurements of tasks running on multicore processors are collected to obtain WCET estimates and provide experimental support evidence of the claims made on the timing analysis technique, i.e., MBPTA.

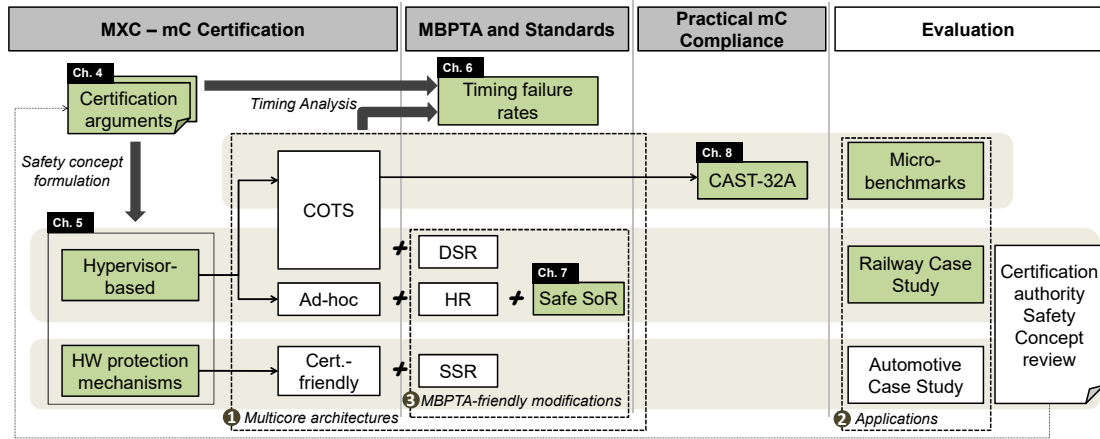


Figure 3.1: Thesis contribution and evaluation dependencies.

3.1 Multicore Architectures

The contributions of this dissertation are evaluated in three alternative multicore architectures to cover a representative enough multicore spectrum: a commercially available multicore, an ad-hoc design and a solution that integrates safety mechanisms on its architecture to foster their certification. The first one, NXP’s P4080, is representative of the complexity in modern multicore architectures relevant for real-time industry. The second, symbolizes an in-house solution that, as opposed to COTS, gives a better insight of processor internals and permits to customize the hardware to implement MBPTA-compliant modifications. Finally, the third platform, AURIX TC27x, integrates many safety mechanisms in its architecture that make it a very promising for systems subject to safety certification constraints.

3.1.1 COTS Multicore: P4080

The NXP’s P4080 ([P4080](#)), whose main elements are sketched in Figure 3.2, embeds eight Power-Architecture e500mc processor cores interconnected through the ‘CoreNet Coherency Fabric’. Each core has private first level instruction (IL1) and data (DL1) caches (32KB each) and a backside L2 cache (128KB). The interconnect is able to perform several concurrent transactions in parallel and manages cache coherency. In addition to the core-local caches, the platform includes a shared L3 on-chip cache (1MB) between the CoreNet and each memory controller, called CoreNet Platform Cache (CPC). The P4080 has two independent DDR3 memory controllers with interleaving support. As there are two memory controllers, there are also two CPC (a total of 2MB) shared among all processors. Additionally, the platform includes multiple peripheral interfaces. All cores have independent boot and reset, as well as, secure boot capability.

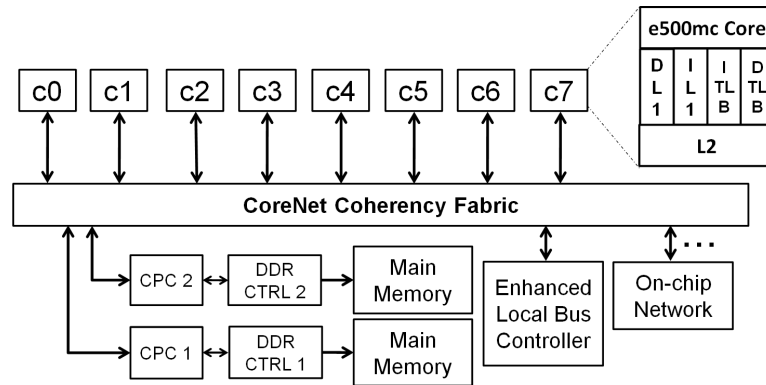


Figure 3.2: High level block diagram of the P4080 (memory hierarchy).

3.1 Multicore Architectures

3.1.2 Ad-hoc Multicore: LEON3-MC

The LEON3-MC is an ad-hoc system-on-chip quad-core architecture designed and prototyped in an FPGA in the scope of the FP7-PROXIMA project. The LEON3-MC (COB16), depicted in Figure 3.3, has four LEON 3 based processor cores of the Sparc v8 architecture interconnected through an AMBA AHB bus. Each processor has an associated L1 cache while the L2 cache and DDR2 memory controller are shared among all cores. The main peripheral interfaces include ethernet controllers, serial communication, timers, interrupt controller and debugging capabilities. The processor has a single clock source.

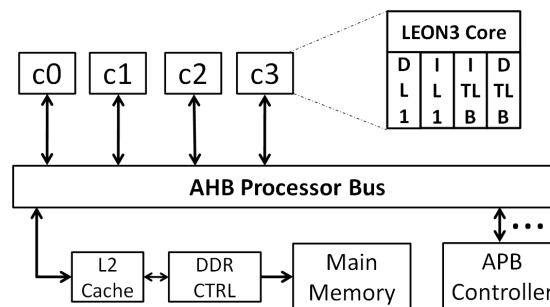


Figure 3.3: High level block diagram of the LEON3-MC (memory hierarchy).

3.1.3 Certification-friendly Multicore: AURIX TC27x

The chosen processor family is the AURIX TC27x COTS multicore provided by Infineon (AURIX). The AURIX TC277, illustrated in Figure 3.4, comprises three cores of the TriCore architecture (plus two additional ones that operate in lockstep mode): one energy-efficient core and two performance-efficient ones. The latter embed high-performance jittery resources such as caches and dynamic branch predictors. All cores are equipped with local scratchpad memories and caches, for both instruction and data, and are connected via a crossbar called Shared Resource Interconnect (SRI) to a common ‘memory system’ comprising a shared SRAM, and program/data Flash memories. Those elements are then connected to medium and low bandwidth peripherals, such as, timers, I/Os and several communication interfaces (e.g., Controller Area Network (CAN)) through the System Peripheral Bus (SPB).

The AURIX is a very promising processor family for CRTES for a number of reasons. It embeds many safety features that makes it suitable to host safety-critical applications up to ASIL-D level easing and reducing certification costs. Moreover, unlike most multicore processors, the AURIX TC27x is designed with

determinism in mind with features that ease the timing analyzability of the platform; for instance, it includes core-local scratchpad memories and it does not share cache memories among the cores.

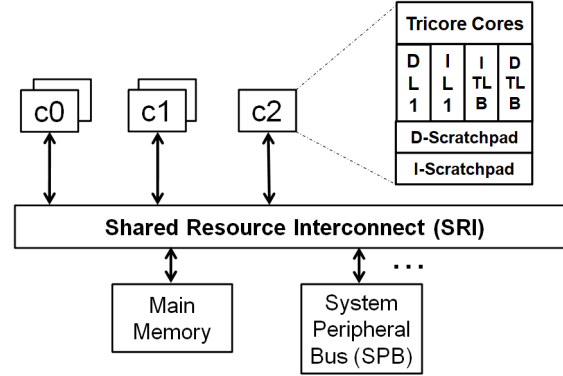


Figure 3.4: High level block diagram of the AURIX TC27x (memory hierarchy).

3.2 Applications

For evaluation purposes we use two different industrial applications from the railway and automotive domains developed in the framework of the FP7 PROXIMA project and microbenchmarks designed to stress specific processor resources. The industrial case studies show the mixed-criticality integration trends in different CRTES domains. Both case studies include safety and non safety-related applications, together with hard real-time restrictions. In addition, each of it is subject to the particular needs of its target domain.

3.2.1 Industrial Railway Application

We have designed a mixed-criticality railway application for both the evaluation of the safety argumentation and to obtain evidence of the industrial application of MBPTA. The case study is comprised of the following on-board train subsystems: (i) safety-critical railway signaling subsystem and (ii) real-time traction control subsystems for the control of the speed/pair of electric motors.

- **European Train Control System (ETCS) on-board Railway Signaling:** The ETCS constitutes the on-board unit of the train signaling system defined in the European Railway Traffic Management System (ERTMS) (ERTMS; GW09), a European Union backed initiative defining a unique

3.2 Applications

train signaling standard throughout Europe. The ETCS is a safety-critical embedded system (Safety Integrity Level (SIL) 4) that protects the train by supervising the traveled distance and speed and by activating the emergency brake if authorized values are exceeded. To this end, it relies on the distance and speed measurements provided by the on-board odometry system, which performs dead reckoning based on a set of diverse sensors (e.g., wheel angular speed encoders, Doppler radars, GPS). The railway infrastructure provides train absolute positions through Eurobalises that are used to correct and recalibrate the odometry subsystem. This is executed in the central safety processing unit of the ETCS, called European Vital Computer (EVC). The EVC executes the safety kernel that includes three main tasks (Figure 3.5):

- Odometry module: is the responsible of estimating a set of parameters based on the information received from the train environment (e.g., estimated train’s position).
 - Emergency module: Controls the emergency braking system.
 - Service module: Controls the service braking system.
- **Railway Traction Subsystem:** It controls the speed/pair of the electric motors varying the switching frequency of a power inverter. For the sake of simplicity and to be able to demonstrate a mixed-criticality setup, the traction subsystem is not considered as safety-related.

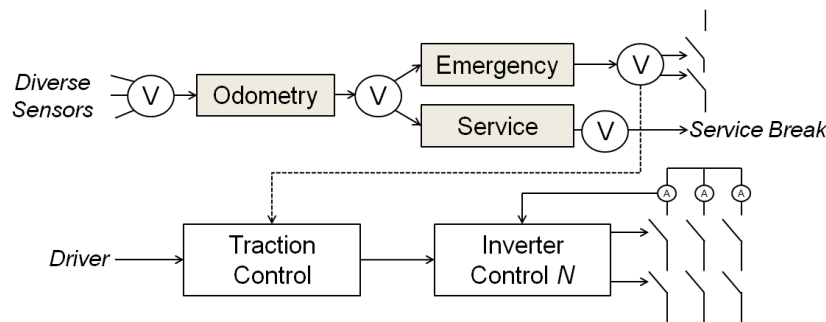


Figure 3.5: Block diagram of the railway case-study application.

The MBPTA technique is applied to a subset of the safety-critical ETCS subsystem when executed on the real-time PikeOS operating system (PikeOS). In particular, it is used to estimate the pWCET of the emergency module. This task is composed of several data dependent paths that, as briefly discussed in the Background Section 2.1.3, have an impact on the timing analysis process. MBPTA provides pWCET estimates that upper-bound the execution time of those paths

exercised with the input vectors. To conduct the measurement-based analysis, a set of input vectors has been defined, which exercises the task at basic block level. These input vectors allow the application of path coverage methods, such as, Extended Path Coverage (MFB+17) over MBPTA. However, the user has commonly means to control those features during timing analysis, such as software code coverage metrics. For simplicity, in this Thesis we exclude the execution path problem from the analysis by studying each path independently by the execution time collection procedure detailed in Subsection 3.3.2.

3.2.2 Industrial Automotive Application

The safety critical application we consider for the automotive safety concept definition is an automotive Cruise Control (CC) system derived from the CONCERTO European research project (CONCERTO). To give the mixed-criticality flavor to the case-study, a non safety-related application is also considered:

- **Cruise Control:** The main job of the CC is to control the speed of a vehicle automatically without human intervention. The system takes control over the throttle of the car to maintain the vehicle's speed as set by the driver. The CC is composed of two safety ECUs that perform the CC functionality by the three tasks illustrated in Figure 3.6:
 - Signal Acquisition ECU: Runs the *Signal Acquisition* task for reading the set of input buttons, pedals and speed sensor and transmits the commands to the engine control ECU.
 - Engine Control ECU: Based on the received commands it sets the required torque value to control car's speed, provide feedback through a lamp and deactivate the CC system when necessary. This is executed by the cruise control *Monitoring* and *Speed Controller* real-time tasks.
- **Power window controller:** The window controller is usually part of the body computer module which implements additional functionalities (e.g., mirror control, central locking). For the sake of simplicity, only the power windows controller is considered as the non safety-related subsystem.

The implementation and timing analysis of this case study were conducted as part of the FP7-PROXIMA project by the University of Padova which was a partner in the project, and therefore are external to this Thesis (KCM+16; AAS+16).

3.2 Applications

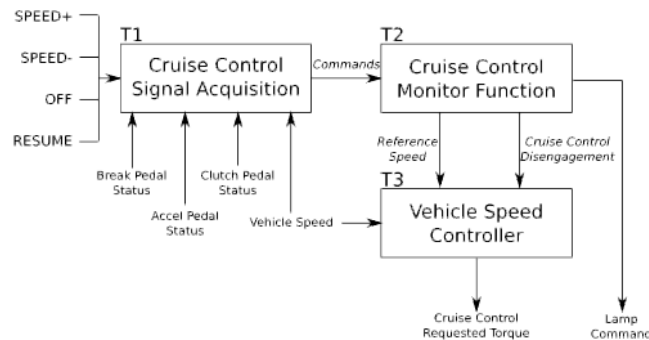


Figure 3.6: Block diagram of the automotive case-study application.

3.2.3 Microbenchmarks

Microbenchmarks (CRV+09; FGQ+12; RGG+12), also known as resource stressing kernels, *rsk*, are a set of specialized programs that put high load on some shared resources and help (among others) determining the existence of some sources of jitter and deriving bounds to their impact on execution time. The microbenchmarks designed in the scope of this dissertation comprise a single loop that contains a number of instructions of the same type, in this case *load* or *store* operations (*OP*), to stress the memory hierarchy as defined in Listing 3.1. The reason to use a loop is to avoid misses in the instruction cache and exercise the desired memory resource long enough to be able to minimize the overhead of instrumenting and measuring the execution time.

```

1 | mr ARRAY, %r0; \n\t           //ARRAY: Address of the array
2 | mr END, %r1; \n\t           //END: End of array (ARRAY + ALLOC_SIZE)
3 |
4 | loop:
5 |   op r2, 0x000(ARRAY);        //op: load or store instructions
6 |   op r2, 0x040(ARRAY);
7 |   op r2, 0x080(ARRAY);
8 |   op r2, 0x0C0(ARRAY);
9 |   op r2, 0x100(ARRAY);
10 |  op r2, 0x140(ARRAY);
11 |  op r2, 0x180(ARRAY);
12 |  op r2, 0x1C0(ARRAY);
13 |  addi ARRAY, ARRAY, sum; //sum: stride * 8
14 |  cmplw ARRAY, END;
15 |  blt loop;

```

Listing 3.1: Exemplary code of memory stressing microbenchmarks.

In the *rsk* initialization, a contiguous section of memory (*array*) of size *ALLOC_SIZE* is allocated. Then, the loop traverses the whole array with a *stride* that determines the distance among two consecutive array accesses. By adjusting the *ALLOC_SIZE* and the *stride* the benchmark is customized to stress different levels of the memory hierarchy. We develop the *rsk* in assembly code to reduce the overhead to a minimum. We then collect end-to-end execution times of *N* executions of this *rsk*.

3.3 Timing Analysis: MBPTA

The MBPTA approach introduced in Section 2.1.3 builds on top of two main pillars: (i) managing the Sources of Jitter (SoJ) during the collection of measurements to ensure that they are representative (upper bound) of systems' operation-time timing behavior; and (ii) applying statistical analysis techniques to those observations to compute a pWCET curve that determines the probability of exceedance of each WCET estimation.

3.3.1 MBPTA-compliant Time Randomized Platforms

As introduced in the Background Section 2.1.3, MBPTA handles the architectural SoJ either by time randomizing them (probabilistically upper-bound) or making them to work in their worst latency (deterministically upper-bound). The modifications required to this end can be introduced following different hardware or software approaches. Earlier work describes how to design and implement these features in an *MBPTA-friendly* platform (KQA+16; KVM+16). In this dissertation we consider four different MBPTA-compliant platforms: (i) LEON3-MC-HR platform, (ii) LEON3-MC-DSR platform, (iii) P4080-DSR platform and (iv) AURIX-SSR platform.

- **LEON3-MC-HR Platform.** Time upper bounding and randomization are applied at hardware level on the LEON3-MC prototyped in an FPGA through the following platform modifications (HAC+17):
 - Cache memories (DL1, IL1 and L2) are time randomized by implementing random placement and replacement policies (KAQ+13a; KAQ+13b; KAQ+14; HAG+16). The former breaks the dependence among memory mapping and cache layouts and the latter releases from the impact of execution history by randomly selecting the victim in every cache miss. In this way, every cacheable memory access will have a hit/miss probability of occurrence in the timing analysis process.

3.3 Timing Analysis: MBPTA

- The Floating Point Unit (FPU) is forced to always work on its worst latency irrespective of which its input values are.
- Bus: The LEON3-MC-HR implements an enhanced random permutation arbitration bus (JKA+14). The basis of the arbitration defines time windows divided in as many slots as cores being arbitrated. This slots are randomly allocated to cores, while ensuring that each core is only assigned one slot within a time window. This random arbitration allows to obtain fully time composable pWCET estimates when analyzing a partition in isolation and making no assumption on the contenders.
- **LEON3-MC-DSR.** The randomization is applied at software level in the baseline LEON3-MC prototype. Instead of modifying the hardware, randomization is introduced at compilation time by using the Dynamic Software Randomization (DSR) technique (KCQ+13; CKW+17). DSR introduces randomization at runtime, by randomly changing the position of memory objects (e.g., functions, basic blocks, data structures) in the main memory across different program runs and adding random padding among them. In this way, objects are randomly mapped into cache lines and this gives each cache layout a probability of occurrence with a similar randomized timing behavior in caches as that obtained with hardware-implemented random placement.
- **P4080-DSR.** The P4080 is a COTS multicore processor and therefore the same DSR technique, ported to PowerPC architecture, is used.
- **AURIX-SSR.** The AURIX is also a COTS multicore commonly used in the automotive domain. In this case, randomization is also introduced at software level but instead of using the dynamic approach of previous platforms, it is done by Static Software Randomization (SSR) by a method called Toolchain-Agnostic Software rAndomization (TASA) (KVM+16; KCM+16). TASA is implemented in the form of a source-to-source compiler that randomly reorders the definition of functions, stack frames and variables in the source code. The approach relies on the observation that compilers generate the elements of the executable (code, data, etc.) in the same order they are encountered in the source file. This makes that the relative location of each element inside the binary determines its placement in main memory, and consequently its mapping in cache. It follows that, with TASA multiple binaries with different (random) memory placements are generated.

3.3.2 Collecting Measurements

To exclude the jitter caused by different execution paths in the program from the analysis, we collect end-to-end execution time measurements of the software Unit of Analysis (UoA) on a per-path basis ($ET(x)$) as illustrated in Figure 3.7. This allows constructing a separate pWCET for each of the traversed paths. Input vectors are therefore designed to trigger a different unique path of the software task under analysis.

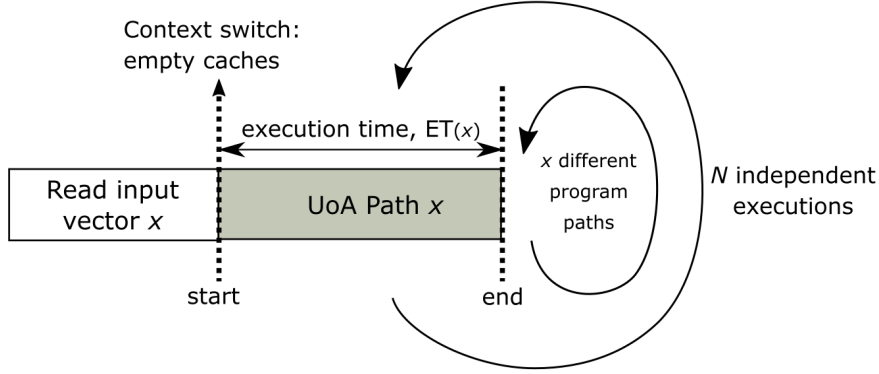


Figure 3.7: Execution time collection for MBPTA.

Similarly, we empty the caches right before executing the unit of analysis to control the initial conditions. Finally, we minimize the impact of the OS by avoiding any context switch and other OS activity during UoA execution. For each path we collect the execution time of N number of end-to-end runs as shown in Figure 3.7, which is usually no less than a hundred and no more than two thousands, which keeps the MBPTA overhead low (APC+17). Execution time measurements are obtained by the introduction of inspection points in the executable code at compile time. During the analysis-time experiments, the specific timestamp at which each point is reached is logged (in this case *start* and *end* points in Figure 3.7). To this end, we use Rapita Verification Suite tool (RVS).

3.3.3 Probabilistic Analysis

The second step of MBPTA, after collecting representative execution time measurements, is to apply probabilistic analysis (EVT (CVQ+13)) to estimate the probability of extreme (timing) events to occur. The utilization of EVT with MBPTA requires that the observed execution times can be described by random variables that are proven independent and identically distributed (i.i.d.). While this property is satisfied by the random timing behavior of MBPTA-friendly platforms (ACQ+13), we apply the i.i.d. tests described in (CSH+12).

Chapter 4

Safety Argument for Multicore Mixed-criticality Systems

Multicore processors are potential candidates to consolidate, in one platform, mixed-criticality applications traditionally distributed in multiple subsystems in a federated paradigm. This is motivated by the increased performance that multicore architectures offer at reduced cost, size, weight and energy consumption. Despite the interest shown by different CRTES industries in multicore architectures, such as automotive ([BK14](#)), avionics ([CAST-32A](#)) or railway ([PikSIL4](#)) among others, their adoption is hindered by the conservative restrictions required to achieve certification. New issues arise from their inherent complexity, unpredictability and limited in-service experience that challenge having sufficient evidence required for certification. In addition, safety standards provide limited guidance for multicore architectures.

This chapter contributes with a safety argumentation for mixed-criticality certification upon multicores. The argument defines safety analysis and guidelines for multicore processors and the partitioned applications running on them based on a separation layer. The argumentation in this chapter focuses on the generic international IEC 61508 standard, as it is written for the functional safety of Electrical/Electronic/Programmable Electronic (E/E/PE) safety-related systems and it is applicable across multiple safety-critical industrial sectors. Several domain-specific functional safety standards are then based on the IEC 61508 standard (cf. [Figure 2.1](#)). Accordingly, the certification metamodel proposed in this chapter is built based on IEC 61508 and then, its application on the reference use case applications of [Chapter 5](#) takes into account the required domain-specific considerations (i.e., EN 5012X and ISO 26262).

This chapter is structured as follows: [Section 4.1](#) introduces the graphical notation used in the rest of this chapter. [Section 4.2](#) introduces the strategy followed to build the safety argument and introduces the top level argument structure.

Section 4.3 briefly describes the safety implications of common-practice federated approaches to later build integrated arguments on top of a well-accepted solution. Section 4.4 and Section 4.5, which are the core of the chapter, define the measures for mixed-criticality integration and emphasize in multicore platforms' implications respectively. Finally, Section 4.6 provides a chapter summary.

4.1 Notation

For building the argumentation metamodel we use the graphical Goal Structuring Notation (GSN) ([GSN11](#)), one of the most commonly used notations to represent safety arguments. The usage of a graphical notation aids in building clearer and better structured arguments than textual narratives. The core elements of GSN, illustrated in Figure 4.1, are the goals or claims represented as rectangles that are then broken down into sub-goals until there is evidence, i.e., a solution, to support them. Strategies provide the reasoning that link goals and sub-goals among them. Some of these claims and strategies hold only under certain assumptions and for a given context or definitions, documented explicitly within the assumption and context elements. Justification symbols provide supplementary information related to the goals or arguments, e.g., a rationale. Some goals and strategies include a hollow diamond symbol to indicate that they are intentionally left undeveloped in the argument. To build compositional arguments with a set of interconnected argument structures, and handle in this way modularity, GSN introduces the module symbol that contains an argument in it and it can be referenced from multiple points in the argument tree by using the away goal symbol.

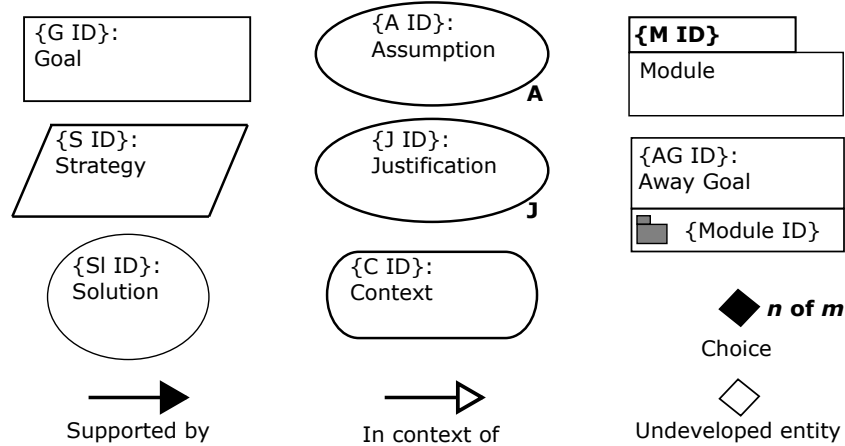


Figure 4.1: Principal GSN elements ([GSN11](#)).

4.2 Overall Safety Argument

This section introduces the overall mixed-criticality multicore safety argumentation in GSN notation using the goal hierarchy depicted in Figure 4.2.

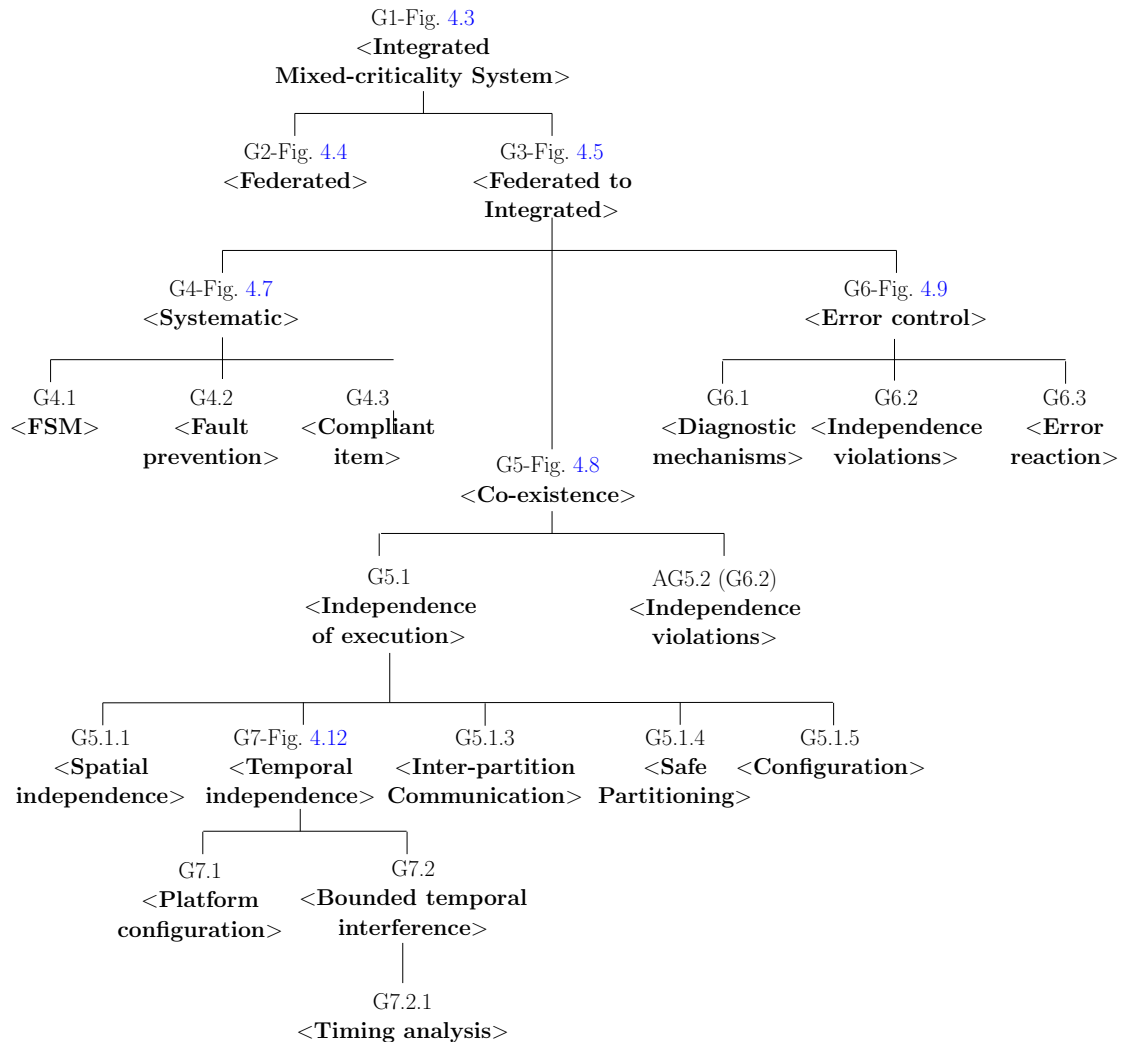


Figure 4.2: Summary of goals in the safety argumentation.

The top level goal of the argumentation is represented in Figure 4.3 and seeks building a safe integrated mixed-criticality system (G1). To bridge the gap between multicore technology and certification, it is based on a certified common-practice federated approach.

G1. The integrated mixed-criticality system is acceptably safe.

G1 can be claimed to be true in the context of the system defined in “C1.1 *System definition*”. The strategy (*S1*) is based on the management of systems hazards, meaning that system hazards are eliminated or sufficiently mitigated up to tolerable rates for the integrity levels assigned to the safety functions (*J1*). To this end, the applicable domain standard shall be defined (*C1.3*) and it is assumed that a hazard analysis has been accomplished (*A1*). The resulting hazard list (*C1.2*) shall be handled by modules *M1.1* and *M1.2* that contain further sub-goals to support *G1*:

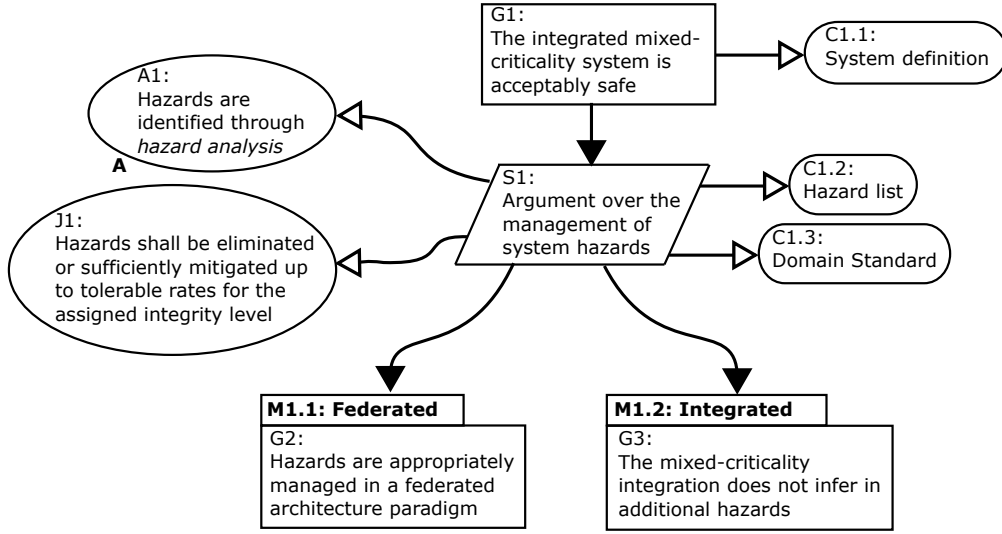


Figure 4.3: Top level argument structure for the mixed-criticality integration based on a certified federated system.

The safety arguments to achieve *G1* are developed in a three-step transformation process:

- M1.1.** includes sub-goal *G2*: “*Hazards are appropriately managed in a federated architecture paradigm*”. *G2* describes the common-practice federated approach to be used as a reference for building the integrated argument (Section 4.3). Software applications of the federated approach are assumed to be certified and reused as compliant items for mixed-criticality integration.
- M1.2.** includes sub-goal *G3*: “*The mixed-criticality integration does not infer in additional hazards*”. *G3* addresses mixed-criticality integration in a top-down approach:

4.3 Common-practice Federated Approach

- Federated to Integrated: the transition from the federated approach to an integrated architecture is evaluated abstracted from the particular platform implementation (either single-core, multiprocessor or multicore) (Section 4.4).
- Multicore Integrated: the mixed-criticality integration is refined to tackle multicore architectures (Section 4.5).

4.3 Common-practice Federated Approach

The common-practice safety case is based on the claim that each safety subsystem is compliant to safety standards in a federated architecture paradigm ($G2.1$). Certifying a federated system with applications of differing safety implications is a widely adopted approach by current safety-related industries. Accordingly, $G2$ is not developed in detail but only the most important aspects are considered in the goal structure of Figure 4.4 to later build integrated arguments on top of a well-accepted approach.

G2. Hazards are appropriately managed in a federated architecture paradigm.

$G2$ (Figure 4.4) is claimed to be true in the context of the federated architecture described in $C2$. A federated architecture is a distributed control system architecture that usually follows a “one function, one computer” paradigm with hierarchical standardized communication buses. Safety-critical subsystems are physically separated from other subsystems and they can be independently certified in compliance to different integrity levels.

In consequence, $G2$ is supported by two sub-goals where the first considers individual certification requirements for each subsystem separately ($S2$) and the second evaluates their interactions:

- G2.1.** Each *subsystem k is compliant with standard*, where k ranges from 1 to N subsystems that perform a safety function. Each safety subsystem k is compliant with the requirements of safety standards for its assigned integrity level ($C2.1.1$).
- Evidence for $G2.1$ is provided by the safety certification (i.e., conformity assessment) of each of the safety subsystems k ($Sl2.1$).
- G2.2.** *Subsystems are physically separated from each other.* This goal shall be intrinsically achieved by the federated architecture paradigm, where each safety application owns private hardware resources and the field-bus provides safe communication (time-triggered communications, bus guardians, etc.).

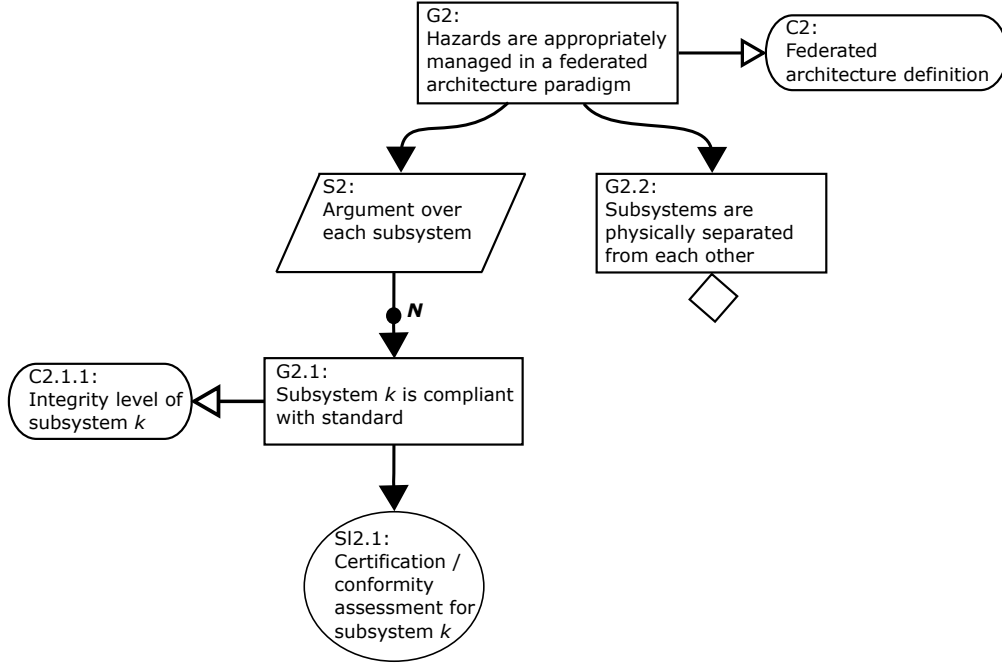


Figure 4.4: Common-practice federated system certification - Goal G2 (M1.1).

4.4 Mixed-criticality Integration

Based on the federated argument of Section 4.2, mixed-criticality integration hazards are managed in Goal *G3*:

G3. Mixed-criticality integration does not infer in additional hazards.

G3 (Figure 4.5) is claimed to be true in the context of the integrated architecture defined in *C3.1*. The strategy followed to support this goal is based on the adoption of safety measures for the management of integration faults (*S3*), which is achieved by modules *M3.1*, *M3.2* and *M3.3*. This strategy assumes that an analysis of the failure modes for partitions is conducted (*A3* and *C3.2*). The individual elements of the goal structure of Figure 4.5 are elaborated in next subsections:

- Section 4.4.1 defines the integrated safety architecture (*C3.1*).
- Section 4.4.2 includes a fault hypothesis and a partition-level Failure Mode and Effect Analysis (FMEA) as part of the failure analysis process (*A3*).
- Each of the modules (*M3.1*, *M3.2* and *M3.3*) for (i) the avoidance of systematic faults during all life-cycle activities (*G4*), (ii) the safe co-existence

4.4 Mixed-criticality Integration

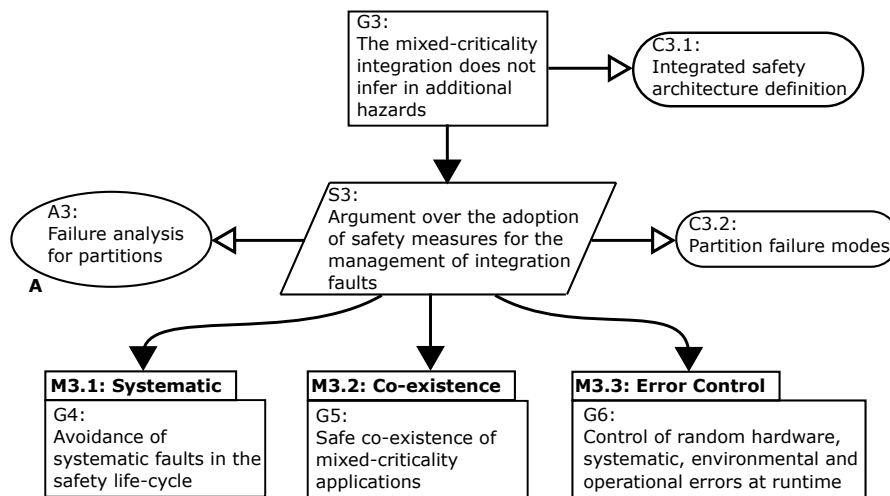


Figure 4.5: Mixed-criticality integration argument - Goal G3 (M1.2).

of mixed-criticality applications ($G5$) and (iii) controlling possible errors at runtime ($G6$) is further detailed within the description of safety measures in Section 4.4.3.

4.4.1 Integrated Safety Architecture

The applications that comprise the federated mixed-criticality system are integrated as multiple partitions in a single platform referred to as Mixed-criticality Central Processing Unit (MxCPU). The safety architecture of the MxCPU is composed of a processor, partitioned software applications and diagnosis (both on-chip and external).

In the IEC 61508 standard the safety architecture is highly dependent on the target integrity level and the Safe Failure Fraction (SFF), i.e., the ratio of safe and dangerous detected failures with respect to all failures. Depending on the Hardware Fault Tolerance (HFT) provided by the architecture, the target SFF varies, which can only be improved either by high design effort or by improving the Diagnostic Coverage (DC) so that the portion of detected dangerous failures is considerably higher than the portion of undetected failures. Commonly, HFT is achieved by hardware node redundancy (several chips) where a HFT of N requires at least $N + 1$ redundant channels. However, implementing this redundancy in a single chip involves many conservative restrictions on the platform design like using separate blocks on substratum for each channel (IEC 61508-2 Annex E) and hence hinders the use of COTS processors for this purpose. For $SIL = 4$ safety functions, on-chip redundancy is not even allowed by IEC 61508. Consequently, in this work the focus is on single-channel integrated approaches without on-chip redundancy

(HFT = 0). For systems requiring a $\text{HFT} \geq 1$ the additional redundant channels shall be implemented on different hardware devices, following a federated approach.

The reference integrated safety architecture (1-channel with separate watchdog) is depicted in Figure 4.6 and builds upon the following assumptions:

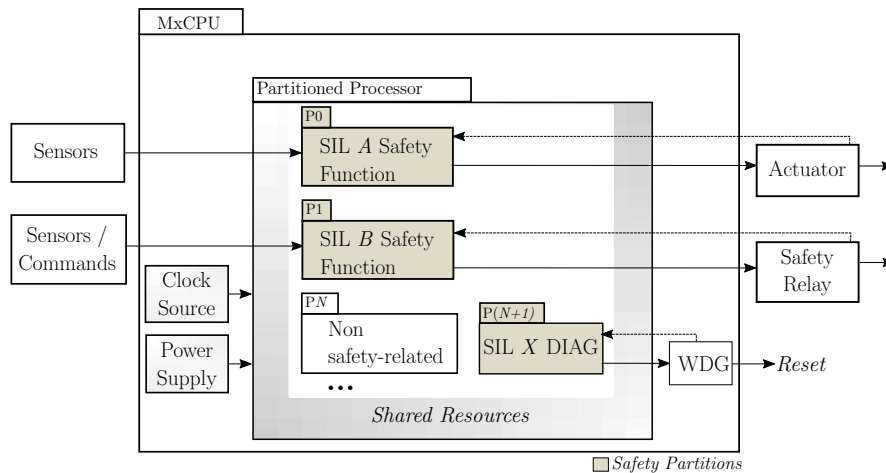


Figure 4.6: Safety architecture for partitioned processor (MxCPU).

Partitioned Processor:

- The partitioned processor conforms a single-channel, i.e., $HFT = 0$ (on-chip redundancy is not considered).
- At this stage, the partitioned processor can either be a single-core processor, a MPSoC, or a multicore.
- The partitioned processor implements a separation layer that guarantees design fault containment among the partitions.

Partitions:

- $N + 1$ partitions are allocated to M cores of the platform, where:
 - N is the number of software subsystems to be integrated in the MxCPU.
 - Extra partition $P(N + 1)$ *DIAG* is defined for diagnosis.
- SIL A , B and X refer to integrity levels, where
 - A and $B \in \{1, 2, 3, 4\}$ for IEC 61508.

4.4 Mixed-criticality Integration

- Different partitions may implement safety functions of different integrity levels (i.e., $A \neq B$).
- $P(N+1)$ DIAG has the maximum integrity level present in the system, i.e., $X = \max(A, B)$.

Watchdog (WDG): The MxCPU integrates a separate watchdog (WDG) external to the processor for diagnosis.

4.4.2 Failure Analysis

The first step for ensuring that $G3$ of the goal structure of Figure 4.5 is fulfilled is to analyze the possible events that may compromise the safety of the system. First, the fault hypothesis for the MxCPU is defined and then an FMEA for partitions is presented.

Fault Hypothesis

The MxCPU fault hypothesis encompasses several safety assumptions to elicit the faults to be addressed in the system design (PGN+14; kop06; PGT+14).

Federated to Integrated - Assumptions. In the transition of a federated system to an integrated approach, we assume that the development and certification of individual components (software applications, hardware platform or components) follow a traditional certification processes:

- Software partitions are compliant items derived from the certified federated applications and hence, are compliant with Functional Safety Management (FSM) and safety life-cycle.
- System, platform (MxCPU) and additional software development is compliant with FSM and safety life-cycle as in the federated approach.
- Off-chip communication is replaced by on-chip inter-partition communication.
- External interfaces are equivalent to the federated approach.
- The system architect is responsible of the safe integration of previous components taking into account safety manuals and applying additional safety measures as required in the standard.

Unit of Failure. The impact of faults in the MxCPU is delimited by the following fault containment regions:

- The partitioned processor forms a single physical FCR. As a result, on-chip redundancy is not considered.
- The external watchdog forms another physical FCR.
- A partition forms a single design FCR (i.e., FCR for software faults).

Failure Modes and Frequency of Failures. The MxCPU, composed of platform, partitioning mechanism (e.g., hypervisor) and external WDG is provided as a compliant item for the appropriate SIL level. It can fail in an arbitrary failure mode and the assumed failure modes and estimated failure rates shall be included in safety manuals.

Failure Mode and Effect Analysis

Taking into account the assumptions in the fault hypothesis, it is assumed that the residual failure rates for the MxCPU are below the acceptable threshold for the particular application domain. Thereby, the FMEA presented in Table 4.1 is abstracted from low level hardware components up to partition level failure modes to emphasize on mixed-criticality integration level failures.

Table 4.1: Simplified Failure Mode and Effect Analysis (FMEA) for partitions.

Failure Mode	Failure Effect	Potential Cause	Failure detection and compensation (MxCPU)
Not Executed	Safety function not performed.	Random HW fault in a platform component (e.g., core)	<Diagnostic mechanisms>
		Design fault in SW (e.g., OS)	<Fault prevention>
		Integration fault (e.g., scheduling)	<Temporal independence>
		Blocking of execution (e.g., contention in shared resource)	<Platform configuration> <Timing Analysis>
Incorrect (functional) Execution	Safety function computes incorrect	Memory / I/O corruption (random)	<Diagnostic mechanisms>

4.4 Mixed-criticality Integration

	outputs.	Memory corruption by contending partition	
		Memory corruption by components with bus master rights	<Spatial indep.> <Configuration>
		I/O corruption by contending partition	
Incorrect (temporal) Execution	Safety function does not compute its outputs on time and may cause dangerous deadline overruns.	Random HW fault in a platform component	<Diagnostic mechanisms>
		Blocking of execution (e.g., contention in shared resource)	<Temporal independence>
		Incorrect allocation of execution time	<Platform Configuration> <Timing Analysis>
		Blocking execution by inter-communication	<Inter-partition communication>
		Integration fault (e.g., scheduling)	<Compliant item> integration tests.
Write' access to non- assigned memory or I/O	Safety peripherals commanded by incorrect partition. Data written by safety partition may be invalidated.	Memory / I/O corruption (random)	<Diagnostic mechanisms>
		Memory corruption	<Spatial indep.> <Configuration>
		Faulty virtualization (wrong address translation)	<Safe partitioning>
Unable to write in assigned memory or I/O	Safety partition outputs not consistent with expected value.	Memory / I/O corruption (random)	<Diagnostic mechanisms>
		Faulty virtualization (wrong address translation)	<Safe partitioning>
		Blocking shared resource	<Temporal independence>
Multiple combinations of previous failure modes	Uncontrolled / unpredictable behavior of safety partition.	Faulty virtualization (privileged partition access)	<Safe partitioning>
		Partition resets core and / or other core(s)	<Safe partitioning> <Spatial independence>
		Uncontrolled interrupts	

Uncontrolled or not
repeatable initialization,
configuration, startup

<Configuration>

4.4.3 Safety Measures

The safety argument of Figure 4.5 is supported by a number of safety measures that aim to reduce the residual risk of integrating mixed-criticality applications to an acceptable level.

G4. Avoidance of systematic faults in the safety life-cycle.

The measures to avoid systematic faults during the different phases of the safety life-cycle are represented in Figure 4.7. The top level goal is divided in three sub-goals based on three requirements of IEC 61508: *S4.1* - complying with FSM requirements in IEC 61508-1 clause 6, *S4.2* - adopting defensive techniques against systematic faults (IEC 61508-3 Annex A) and *S4.3* - integrating compliant items appropriately (IEC 61508-2 and -3 Annex D).

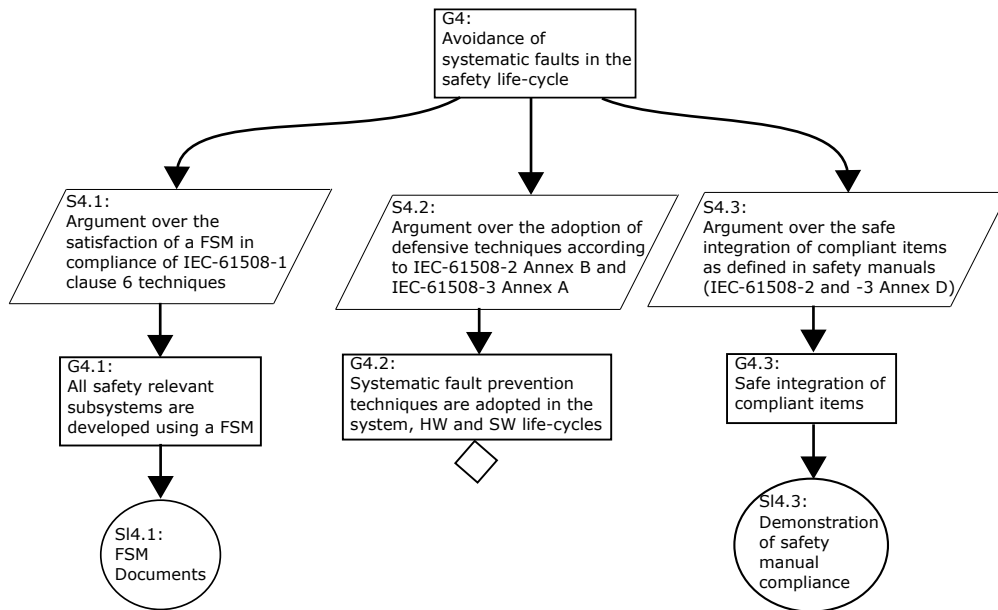


Figure 4.7: Argument for the avoidance of systematic faults - Goal G4 (M3.1).

G4.1. *All safety relevant subsystems are developed using a <FSM>, including system, platform and partitions (IEC 61508-1 Clause 6 compliant). Documentation evidence shall be made available (SI4.1).*

4.4 Mixed-criticality Integration

G4.2. *Systematic <Fault prevention> techniques are adopted in system, HW and SW life-cycles.* This reduces the probability of introducing systematic faults during system life-cycle (IEC 61508-2 Annex B), including software (e.g., IEC 61508-3 Table A.4 Defensive Programming) and hardware (IEC 61508-2 Table F.1) development.

G4.3. *Safe integration of <Compliant items> according to hardware (IEC 61508-2 Annex D) and software (IEC 61508-3 Annex D) safety manuals.* The system architect is responsible for analyzing safety manuals of compliant items and to define their safe integration into the system. Adherence to safety manuals shall be demonstrated (*Sl4.3*).

G5. Safe co-existence of mixed-criticality applications.

Module *M3.2* is developed in Figure 4.8, where goal *G5* aims to achieve the composable certification of each application according to its criticality level as defined in IEC 61508-3 paragraph 7.4.2.9 (*S5*).

The argumentation for *G5* aims to provide enough evidence of independence of execution (*G5.1*) among the applications of different criticality levels or, alternatively, guaranteeing that any violation of independence is controlled (*AG5.2*).

G5.1. *There is <Independence of execution> among applications.* Independence of execution is pursued based on partitioning techniques described in IEC 61508-3 Annex F (*S5.1*):

G5.1.1. *There is <Spatial independence> among partitions.* Spatial independence ensures that one partition does not modify the data or I/Os of another partition, either when the partitions run in the same core or in different processing units of a MPSoC or multicore:

- Memory: Each partition is assigned a private address range (virtual or physical) protected by means of MPU / MMU.
- I/O: Each safety partition has exclusive access rights to shared peripheral devices.

M5.1.2. *There is <Temporal independence> among partitions.* Temporal independence guarantees that one partition does not exceed its timing deadline due to the effect of other partitions, either when the partitions are allocated to the same core, or running simultaneously in different processing units of a MPSoC or multicore:

- Same core: Partitions allocated to the same core share processor time in a statically defined cyclic scheduling basis supported by WCET analysis.

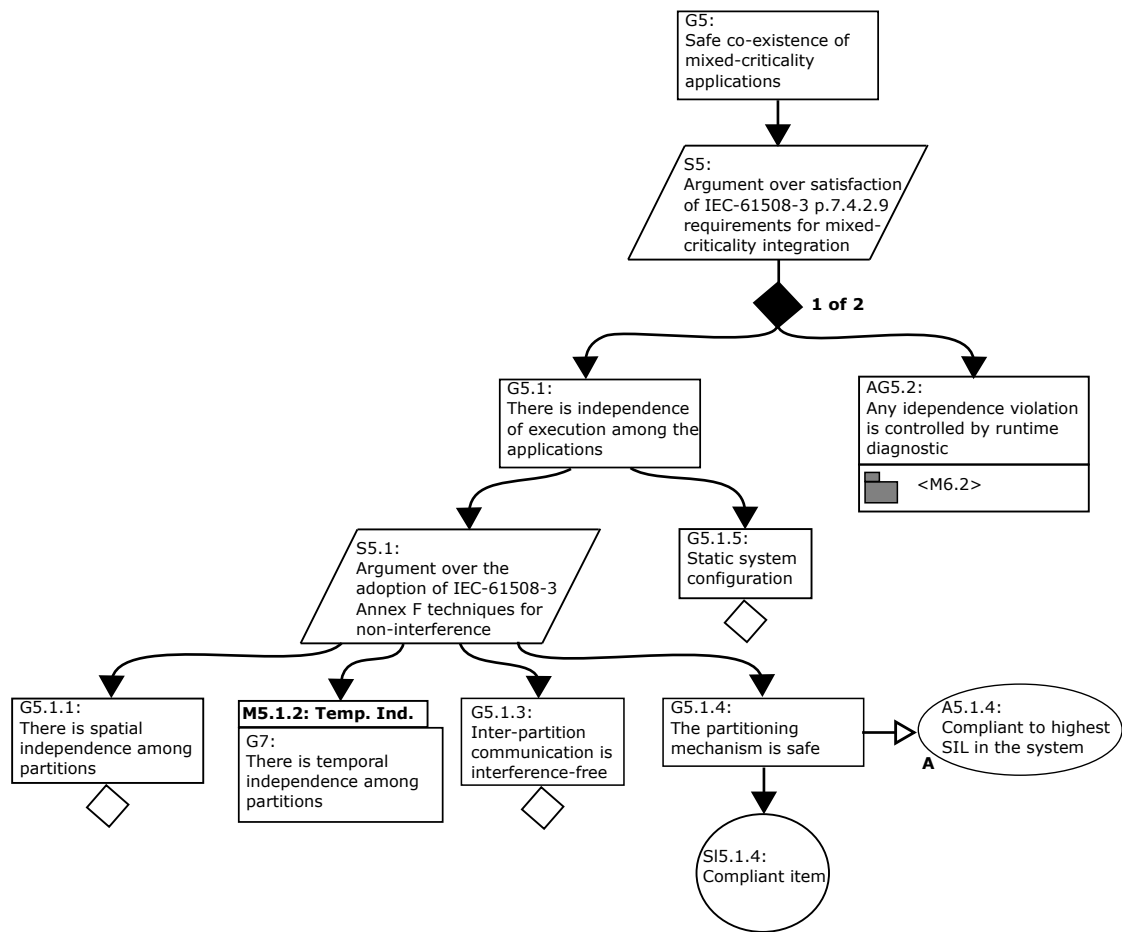


Figure 4.8: Argument for the co-existence of mixed-criticality applications - Goal G5 (M3.2).

- Different cores: For partitions simultaneously executing on different cores and using shared resources, the scheduling shall preserve all timing deadlines. This module, based on the analysis of time interferences (multicore contention), is explained in more detail in *G7* (Figure 4.12).

G5.1.3. *<Inter-partition communication> is interference-free.* Data communication interfaces shall be unidirectional and they shall guarantee that data transmission does not block or cause interferences in the execution of partitions.

G5.1.4. *The partitioning mechanism is safe (<Safe partitioning>).* The mechanisms used to enforce partitions (e.g., hypervisor) shall be certifiable (*SI5.1.4.* compliant item) according to the highest in-

4.4 Mixed-criticality Integration

tegrity level among all partitions in the system (A5.1.4).

G5.1.5. *Static system <Configuration>* is defined (by the system architect) during design stage with qualified tools. It includes configuration features, such as, memory and I/O association to partitions, partition to core allocation, scheduling and inter-partition communication channels. System configuration is protected against unintended runtime modification.

G5.2. *Any <Independence violation> is controlled by runtime diagnostic measures* defined in the away goal AG5.2 that points to module M6.2 of goal structure of Figure 4.9. In the event of detecting an interference, the system is turned into its safe state. As a consequence, controlled interferences do not jeopardize system safety but its availability.

G6. Control of random hardware, systematic, environmental and operational errors at runtime.

At runtime, the effects of random hardware, systematic, environmental and operational errors shall be controlled. To this end, as depicted in Figure 4.9, error detection diagnostic measures are implemented (S6.1), independence violations are controlled (S6.2) and safe system reactions to errors are defined (6.3) assuming that the system is fail safe (A6.3) and that it therefore disposes of a safe state defined in C6.3.

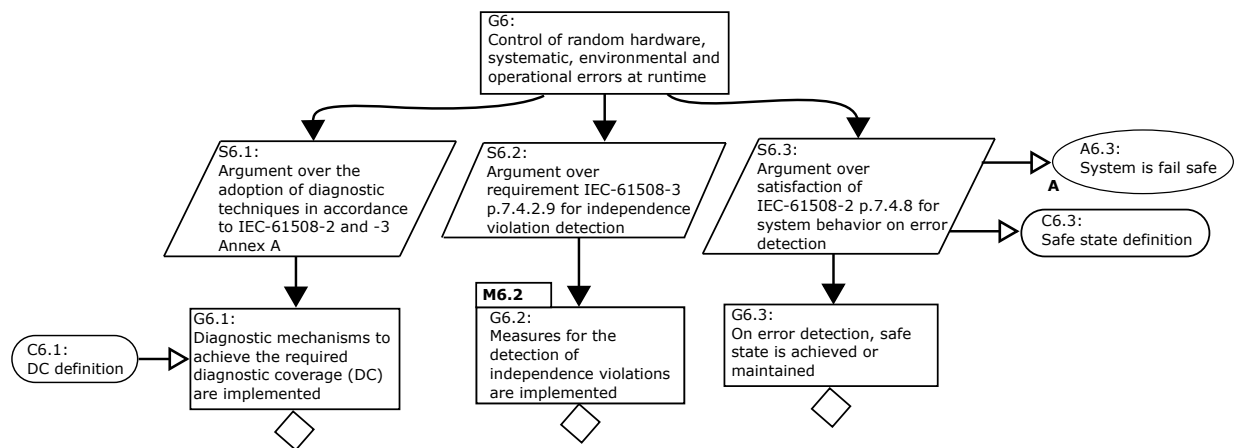


Figure 4.9: Argument for the runtime control of errors - Goal G6 (M3.3).

G6.1. *<Diagnostic mechanisms> to achieve the required diagnostic coverage (DC) are implemented* for runtime error detection. The particular measures shall be selected from IEC 61508-2 and -3 Annex A according to the required diagnostic coverage defined in C6.1. They include:

- Autonomous hardware diagnostics: the MxCPU includes autonomous diagnostic mechanisms (e.g., Power Failure Monitor (PFM) IEC 61508-2 Table A.9).
- Software-commanded diagnostics: the MxCPU includes hardware diagnostic components to be commanded by software including features for the diagnostic of independence violations (e.g., IEC 61508-2 Table A.10 watchdog, MMU/MPU). These mechanisms are implemented in the DIAG partition.
- Platform independent diagnostics: additional diagnostics for hardware components (e.g., IEC 61508-2 Table A.7 Input comparison voting) and software applications (e.g., IEC 61508-3 Table A.2 fault detection by software diversity).
- MxCPU Diagnostics: system diagnostics external to the partitioned processor, e.g., off-chip redundancy with majority voter (IEC 61508-2 Table A.4) or external watchdog for temporal and logical monitoring (IEC 61508-2 Table A.11).

G6.2. *Measures for the detection of <Independence violations> are implemented by hardware diagnostic mechanisms (e.g., MPU, watchdog) and by the DIAG partition that supervises the correct temporal behavior of each partition and handles the external WDG in case of execution time exceedance events.*

G6.3. *On error detection, safe state is achieved or maintained (<Error reaction>). System reactions are defined to control the effects of faults in case of error detection. The safe state defined in C6.3 of Figure 4.9 must be reached or maintained by the safety function or safety mechanisms. In the reference architecture of Figure 4.6 the safe state is reached by the safety chain illustrated in Figure 4.10 either by: (i) the safety function, (ii) safety measures after detecting an error by diagnostics (e.g., WDG reset) or (iii) inherently (e.g., power down).*

- DIAG partition handles the errors detected by the diagnostic mechanisms in the platform or partitions and controls the external WDG. In case of error detection, it does not refresh WDG.
- If the WDG is not refreshed in time, it reaches time-out and resets the platform. Platform reset de-energizes safety-relays achieving safe state. In addition, the watchdog detects timing errors in the processor or faults affecting to the DIAG partition.
- The system is *inherent fail safe*, i.e., falls into the safe state automatically during system initialization, power off, shut down and reset states (e.g., digital outputs are de-energized by default).

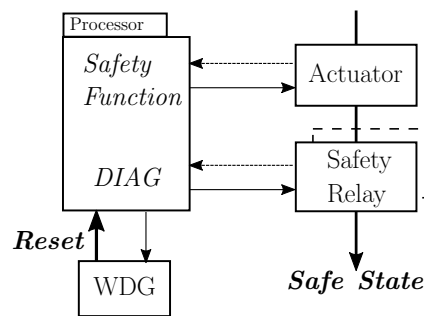


Figure 4.10: Safety chain for safe state activation.

4.5 Multicore Integration

In multicore architectures, running software applications in parallel makes that the access of applications to platform resources is not exclusive anymore. As a result, it is required to analyze the architecture to determine the impact of the contention in shared hardware resources and their effects on execution time.

4.5.1 Multicore Safety Architecture

The architecture of Figure 4.6 is adapted to include the features of a generic multicore. The multicore safety architecture with main hardware resources of relevance is depicted in Figure 4.11:

Multicore Processor:

- The multicore processor conforms a single-channel, i.e., $HFT = 0$ (1-Channel with separate watchdog).
- M is the number of cores of the multicore platform.
- For systems requiring $HFT > 0$, off-chip redundancy (external to the multicore processor) is required.
- The processor implements a separation layer that guarantees design fault containment among the partitions.

Partitions:

- $N + 1$ partitions are allocated to the M cores of the multicore:
 - Partitions can be exclusively allocated to a core ($P0 \rightarrow c0$ and $P1 \rightarrow c1$), or

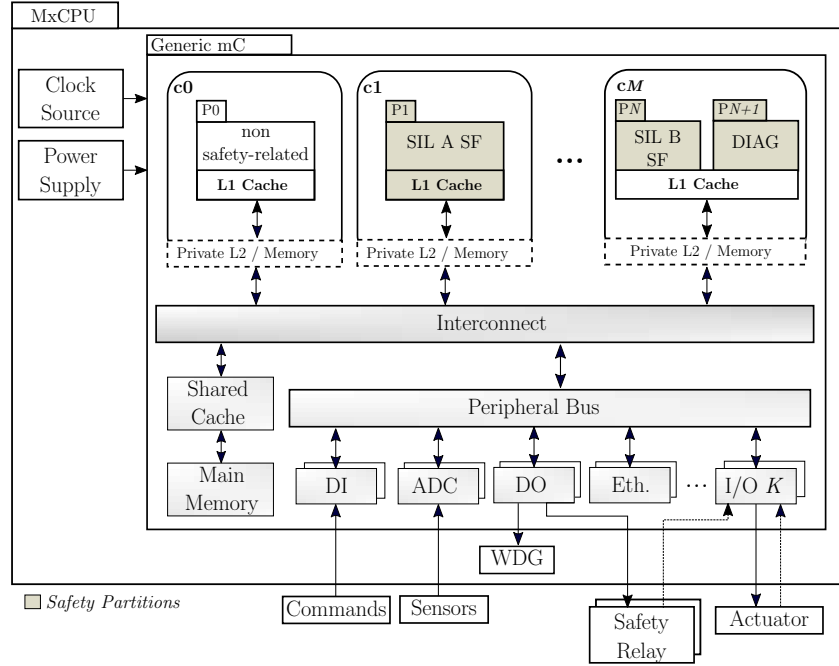


Figure 4.11: Reference safety architecture for generic multicore (MxCPU).

- various partitions to the same core (PN and $P(N + 1) \rightarrow cM$).
- SIL A , B and X refer to integrity levels, where $X = \max(A, B)$.

Watchdog (WDG): The MxCPU integrates a separate watchdog (WDG) external to the processor for diagnosis.

4.5.2 Independence in Multicore Architectures

In the transition from the integrated approach to multicore architectures, the main challenge rests in preserving independence guarantees ($G5$ of Figure 4.8). This is explained by the fact that, while spatial separation can still be attained by state of the art mechanisms (MMU/MPU), temporal independence is jeopardized by the jittery resources present in multicores that cause hardly predictable execution time variations and the simultaneous access to shared resources by multiple software applications causing multicore contention. This section defines module $M5.1.2$ of goal $G5$ (Figure 4.8) for assessing temporal independence (bounded interference) in multicores.

G7. There is temporal independence among partitions.

This goal is achieved by ensuring that software partitions will not fail in the temporal domain (i.e., miss a timing deadline) due to the timing interferences caused by other partitions running simultaneously in other cores in multicore architectures. This requirement involves minimizing the possible interferences in the platform configuration (G7.1) and defining a static schedule of partitions (S7) that considers the worst possible impact of interferences (e.g., shared resources) (G7.2).

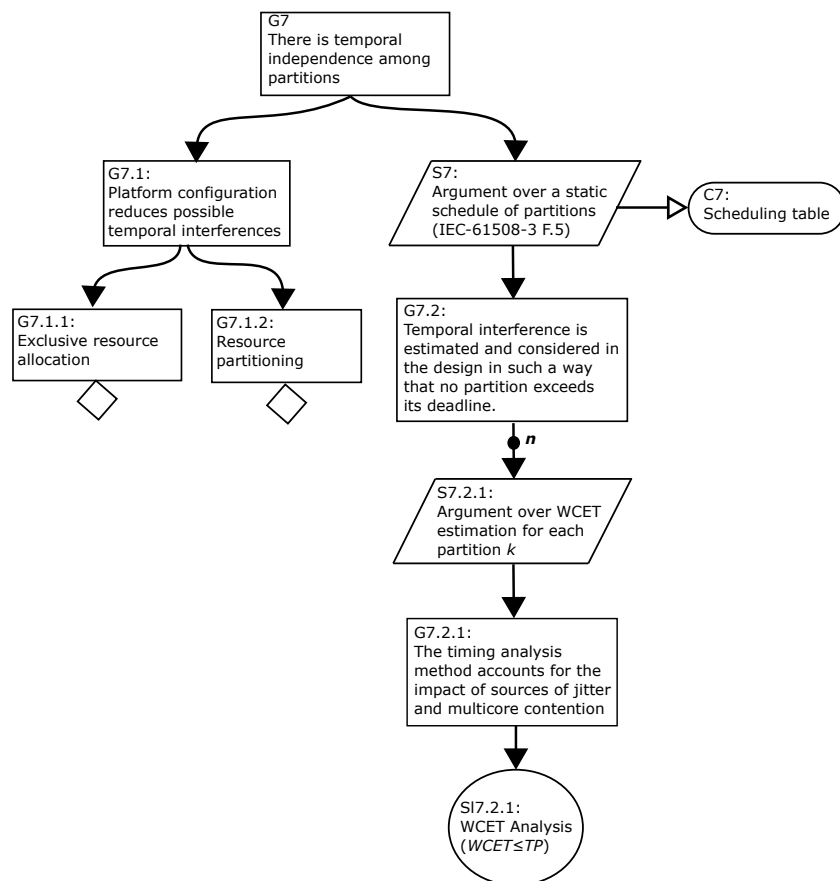


Figure 4.12: Multicore temporal independence argument - Goal G7 (M5.1.2).

G7.1. <Platform configuration> reduces possible temporal interferences.

The aspects impacting execution time shall be studied and defined:

- G7.1.1.** Resource allocation: Partition to core allocation, exclusive use of peripherals, bus access priorities, resource configuration.
- G7.1.2.** Resource partitioning: shared cached partitioning, memory space partitioning.

G7.2. *Temporal interference is estimated and considered in the design in such a way that no partition exceeds its deadline (<**Bounded temporal interferences**>).* Considering the impact of <**Bounded temporal interferences**> in the design guarantees that the static scheduling table (C7) defined by the system architect is suitable to run all partitions without exceeding deadlines. This goal is supported by the WCET analysis of each partition k (S7.2.1), where k ranges from 1 to n partitions:

G7.2.1. *The <**Timing analysis**> method accounts for the impact of sources of jitter and multicore contention.* This claim shall be evidenced by demonstrating that computed WCET is safe and always below the budget assigned to the time partition (TP), i.e., $WCET \leq TP$ (Sl7.2.1).

In addition, temporal independence shall be preserved by the control of <**Independence violations**> as explained in G5 (Figure 4.8). If WCET estimates are too optimistic, when exceeded the respective guard (WDG) protects the system switching to a safe state at the expense of compromising system availability.

4.6 Summary

The work in this chapter reduces the big conceptual gap between multicore processors and certification. In this chapter we thoroughly review IEC 61508 requirements and propose a set of guidelines, in the form of GSN argumentation, to comply with them in a multicore based mixed-criticality architecture. To this end, the safety argumentation defines IEC 61508 based safety measures at different levels of abstraction. First, we define a reference safety ECU incorporating a multicore processor and safety mechanisms for diagnosis purposes. Then, we focus on the safety analysis at software partition level by the definition of a generic FMEA that defines the failure modes for partitions. These failure modes are handled by separation mechanisms that seek to guarantee the independence of execution among partitions. All this is complemented by functional safety management throughout the development process.

In summary, we propose a three-step incremental adaptation to multicores by taking current federated and single-core integrated practices as a baseline. First, an overview of the common-practice federated approach is provided. Then, the integration of the federated subsystems into a single platform is evaluated. Finally, the last refinement step focuses on multicore specific challenges which are mainly related to the platform configuration (i.e., resource allocation) and the timing predictability of the platform.

Chapter 5

Multicore Mixed-criticality Safety Concept

Safety certification is a long process that includes several interactions with a certification body until the system is regarded as safe for its purpose. An important aspect of the certification is the review of specific documentation generated throughout the development process. This documentation often includes the definition of a safety concept with safety analysis, assumptions, requirements and measures to achieve the required level of safety. A preliminary concept review is usually a recommendable industrial practice to assess the design and detect any failure, inconsistency or limitation at the early phases of the life-cycle.

This chapter defines two multicore mixed-criticality safety concepts for the railway and automotive domains based on the safety argumentation metamodel defined in Chapter 4. The two safety concepts rely on alternative solutions and different multicore architectures. The first one, deals with multicore integration requirements by the adoption of a certified hypervisor that provides the required partitioning over an ad-hoc and a COTS multicore architecture. On the contrary, the second safety concept is build upon a certification-friendly multicore for the automotive domain that integrates a set of hardware protection mechanisms. For evaluation purposes, both safety concepts have been reviewed and judged as suitable by an external certification body (TÜV Rheinland). The positive result reflects that it is feasible to achieve multicore certification according to IEC 61508 if several assumptions with respect to the timing analyzability of the multicore processors hold valid.

The rest of this chapter is structured as follows. Section 5.1 provides the definition of the terms used in this Chapter. Section 5.2 presents the railway safety concept over the ad-hoc LEON3-MC (COB16) and COTS P4080 (P4080) multicores. Likewise, Section 5.3 presents the automotive safety concept on the AURIX processor (AURIX). Finally, Section 5.4 presents a summary of the chapter.

5.1 Glossary

1oo1 “1 out of 1” single channel architecture without redundancy where the node shall be operational to preserve safety function (HFT=0).

2oo3 “2 out of 3” architecture, where at least two redundant nodes out of three shall be operational to preserve safety function (HFT=1).

Address Protection Sets (APS) Access protected instruction and data areas in the core-local memories of the AURIX TC27X processor.

ASIL Automotive Safety Integrity Level ranging from A to D ([ISO26262](#)).

Compliant item An element or subsystem on which a claim is being made with respect to its compliance to a safety standard ([IEC61508](#); [EN50126](#)).

Composite fail safety Safety-related functions performed by at least two items independent from each other ([EN50126](#)).

CoreNet Platform Cache (CPC) Last level shared cache (L3) on the P4080.

Failure in Time (FIT) 1 failure per 10^9 hours of operation ($10^{-9}h^{-1}$).

Fault Containment Region (FCR) The boundaries of an immediate failure impact.

Fault Tolerant Time Interval (FTTI) Time-span in which a fault can be present before a hazardous event occurs (equivalent to PST) ([ISO26262](#)).

Functional Safety Management (FSM) Organizational and failure-avoidance measures.

Hardware Fault Tolerance (HFT) Number of faults that don’t cause the loss of the safety function.

Health monitoring Set of functions that monitor the state of (virtualized) applications.

MxCPU Mixed-criticality Central Processing Unit, composed of platform, partitioning mechanism and external *WDG*.

PAMU Peripheral Access Management Unit in the P4080 processor.

Process Safety Time (PST) The time by which an action has to be completed to prevent a hazardous event occurring ([IEC61508](#)).

Protected Memory Region (PMR) A specific memory range with exclusive write permissions in the shared SRAM of the AURIX TC27X processor.

Qualified tool Software tool that provides the required level of reliance.

RAMS Reliability, Availability, Maintainability and Safety.

Safety Communication Layer (SCL) Services and protocol for safety communications.

Safety Element out of Context (SEooC) Equivalent to *compliant item* for the automotive domain ([ISO26262](#)).

Safety goal Top-level safety requirement in the automotive domain ([ISO26262](#)).

Safety manual Document that provides all the information relating to the functional safety of a *compliant item*.

SIL Safety Integrity Level ranging from 1 to 4 ([IEC61508](#); [EN50126](#)).

Triple Module Redundancy (TMR) Three implementations of the same logic with voting.

Watchdog (WDG) External timing element to monitor computer’s behavior.

5.2 Railway Safety Concept

This section defines a safety concept for the industrial mixed-criticality railway system, for its evaluation and review with a certification authority. The case study comprise the two self-contained subsystems introduced in Section 3.2.1: an ETCS signaling subsystem that performs *SIL* 4 safety functionality and traction subsystems that control train's motors. The safety concept builds on the generic IEC 61508 standard, inline with the safety argumentation presented in Chapter 4. However, where required, references are made to railway domain standards (EN 5012x).

5.2.1 Safety Requirements

Table 5.1 summarizes the most representative safety requirements for the railway ETCS system (SR_RE), which are extracted from the ETCS standard (ERTMS).

Table 5.1: Main safety requirements for the EVC.

ID	Requirement
SR_RE_1.A	The safety function "Speed and distance supervision" supervises that the 'train speed' does not exceed the 'maximum authorized train speed' and that the 'train traveled distance' does not exceed the Movement Authority (MA).
SR_RE_1.B	The safety function "Speed and distance supervision" must be provided with a <i>SIL</i> 4 integrity level (EN 50126).
SR_RE_1.1	The safety function "Speed supervision" ensures that the 'train speed' does not exceed the 'maximum authorized train speed'. If exceeded 'emergency brake' safety function is activated.
SR_RE_1.2	The safety function "Distance supervision" ensures that the 'train traveled distance' does not exceed the MA. If exceeded 'emergency brake' safety function is activated.
SR_RE_1.3	The safety function "Emergency brake" activates the 'safe state'
SR_RE_1.4	The 'safe state' is the de-energization of output 'safety-relay(s)'.
SR_RE_1.5	The <i>Process Safety Time (PST)</i> is 1 second.

The ETCS standard (ERTMS) provides an extensive set of documents that specify in detail the system interoperability and safety requirements. These documents could be directly used to identify safety requirements as needed in phase 4 (System Requirements) of EN 50126 with no need to perform a risk analysis. The railway case study is a simplification of this standard.

5.2.2 Federated Safety Concept

On-board railway subsystems are traditionally implemented in a federated approach as sketched in Figure 5.1. In order to safely perform the *SIL* 4 safety requirement SR_RE.1.A, the central processing unit of the ETCS subsystem is implemented as *composite fail safety* by means of *Triple Module Redundancy (TMR)* computing nodes (EVC_TMR). The TMR safety architecture is achieved by three parallel implementations of the same EVC logic and a comparison of the outputs. This architecture allows to identify any discrepancies among the replicated nodes.

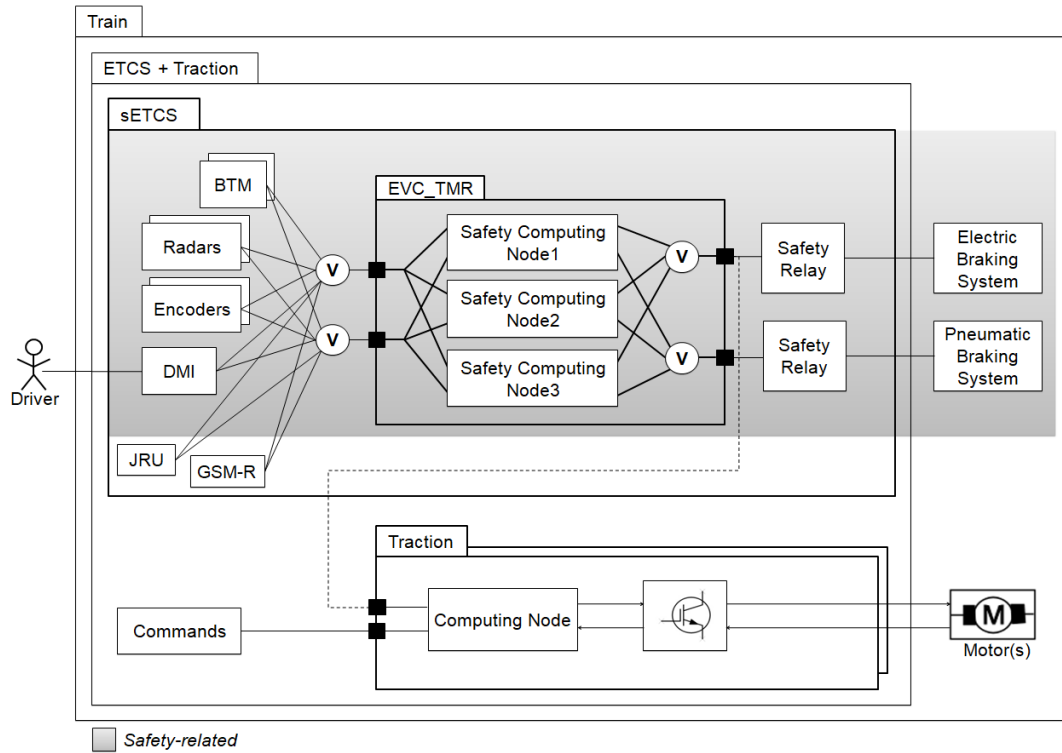


Figure 5.1: Railway case study context diagram.

Each computing node has an instance of the safety software application and satisfies the following safety features:

- A. *Life-cycle and development tools*: The development of safety-related nodes is compliant with safety life-cycle (EN 50126 clause 5.2 / IEC 61508-1 clause 7) and corresponding *FSM* (EN 50126 clause 5 for *RAMS* management / IEC 61508-1 clause 6).
 - A.1. Safety-related nodes are developed using *qualified tools* and compilers (for up to *SIL* 4 EN 50128/EN 50129 and IEC 61508 *SIL* 3).

5.2 Railway Safety Concept

- A.2. The system architect takes reasonable measures to develop a safe product taking into consideration the *safety manuals* of all compliant items (as requested in 4.7.6c EN 50126 / IEC 61508 parts 2 and 3 Annex D).
- B. *Architecture: 2oo3* safety architecture composed of *TMR* computing nodes compliant with *composite fail safety* technique (EN 50129 clause B.3.1 / IEC 61508-2 Table 3 $HFT = 1$).
 - B.1. The system achieves EN 50128/EN 50129 and IEC 61508 *SIL* 4 in *TMR* architecture.
 - B.2. Platform *compliant item*: each computing node, composed of hardware and associated platform software is compliant with EN 5012X / IEC 61508 *SIL* 3 (see A.2).
 - B.3. Each computing node executes a software application instance of the EVC (see C).
 - B.4. The internal communication among EVC nodes is implemented by a ‘safe communication’ protocol (see D).
 - B.5. Safety-related subsystems are physically separated (in different ECUs) from non safety-related applications.
- C. *Software Application*:
 - C.1. EN 50128 *SIL* 4 (IEC 61508 *SIL* 3) life-cycle and FSM compliant software development for EVC computing nodes.
 - C.2. Safety-related inputs, subsystems and outputs are managed exclusively by associated safety-related software tasks.
 - C.3. After start-up, the safety software application commands digital-outputs according to ‘emergency brake’ command (emergency brake activation corresponds to digital-outputs deactivation).
- D. *Safe Communication*: Safe communication protocol such as compliant TTEthernet ([KAG+05](#); [Ste08](#)) or black channel safety communication that provides equivalent safety techniques and attributes of interest:
 - D.1. Safety-related computing nodes integrate a *Safety Communication Layer (SCL)* provided as *SIL* 3 compliant item.
 - D.2. When used in redundant star topology with *TMR* system the SCL can reach up to *SIL* 4.

- D.3. The *SCL* ensures safe data communication among communicating computing nodes, and provides diagnosis to detect all possible failure modes of a black channel communication: frame corrupted, incorrect order of frame, frame outside temporal constraints and frame lost.
- D.4. A global notion of time is established by means of a transparent global clock synchronization for the EVC and safety-related subsystems.
- D.5. *SCL* associated tools are qualified tools, for up to *SIL* 4 (EN 50128 / EN 50129) and IEC 61508 *SIL* 3.

E. *External Interfaces:*

- E.1. Each computing node manages two redundant digital outputs (at least *SIL* 2) to be connected to two external redundant safety-relays that perform 2oo3 voting.
- E.2. The safety-relays participate on the “Emergency brake” safety function (requirement SR_RE_1_3) and safe state activation (requirement SR_RE_1_4) by controlling two diverse braking systems, an electric and a pneumatic one.
- E.3. The fail safe state of digital outputs is de-energized (see F).
- E.4. The default state of digital outputs is de-energized, i.e., during no-power and start-up (inherent fail safe state).
- E.5. After start-up the safety software application commands digital-outputs (activation and de-activation).
- E.6. Each external safety-relay provides an output that represents the output state of the safety-relay, input to the EVC, used for diagnosis purposes.

F. *Safe state:*

- F.1. Safe state is achieved by means of de-energization of safety digital-outputs connected to external safety-relays.
- F.2. If the system, hardware or software diagnosis detects a ‘major’ or ‘significant’ error the safe state must be reached within the *PST* (requirement SR_RE_1_5).

G. *Diagnosis:*

- G.1. At system level the EVC implements diagnosis techniques according to EN 50126, EN 50128, EN 50129 and IEC 61508. The EVC supports up to *SIL* 4 level with *HFT* = 1, therefore, at least a $DC > 99\%$ is required according to IEC 61508-2.

5.2 Railway Safety Concept

- G.2. At node level, each safety computing node is a *SIL* 3 IEC 61508 compliant item with $HFT = 0$, and therefore, requires a $DC > 99\%$ for memories, power supply, clock, program sequence, I/Os, communication, etc. (EN 50129 Annex E and IEC 61508-2 Tables A.2-A.14).
- G.3. The safety-related software implements safety life-cycle related techniques (e.g., IEC 61508-3 Table A.4 defensive programming), performs reciprocal comparison diagnosis (“monitored redundancy” of the triplicated application, IEC 61508-3 Table A.2) and complements/uses diagnosis techniques provided by the compliant platform, (e.g., refresh watchdog, monitor external safety-relays).
- G.4. The system architect implements additional measures taking into consideration the *safety manuals* of all compliant items (as requested 4.7.6c EN 50126, IEC 61508-2 and 3 Annex D).

5.2.3 Integrated Node Architecture

Non safety-related traction computing nodes are integrated into one of the triplicated EVC computing nodes following an integrated architectural approach based on mixed-criticality. At system level the *TMR* safety architecture remains in a federated fashion (with each safety node in a different computing platform). Nevertheless, the node level integration of traction nodes together with one of the EVC replicas benefits from the advantages of integrated mixed-criticality systems.

Integrated Safety Architecture

The different functionalities that compose the railway case study are allocated into different partitions of the Mixed-Criticality Node Safety CPU (*MxCPU*) in the architecture depicted in Figure 5.2 and described below:

- *MxCPU*: Composed of partitioned processor, hypervisor and external *WDG*.
- Partitions: Subsystems of the federated approach are segregated into different software partitions based on criticality.
 - The ETCS is segregated into three different partitions:
 - * *P1-ODOMETRY* estimates the traveling distance and speed of the train. As the main safety function “Speed and distance supervision” (SR_RE_1_A) relies on the parameters computed by the odometry, this partition is *SIL* 4.
 - * *P2-ETCS EVC (Safe)* performs the *SIL* 4 “Speed and distance supervision” safety function (SR_RE_1_A).

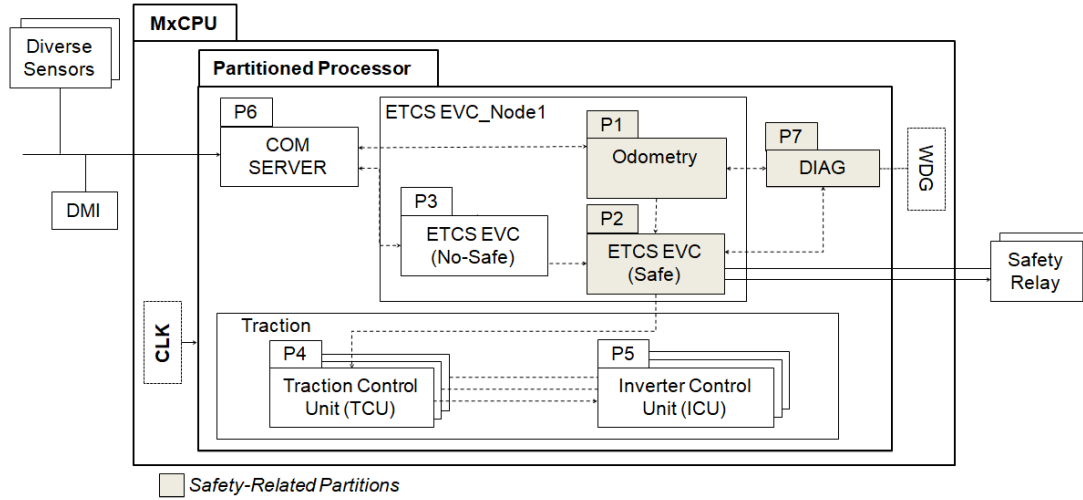


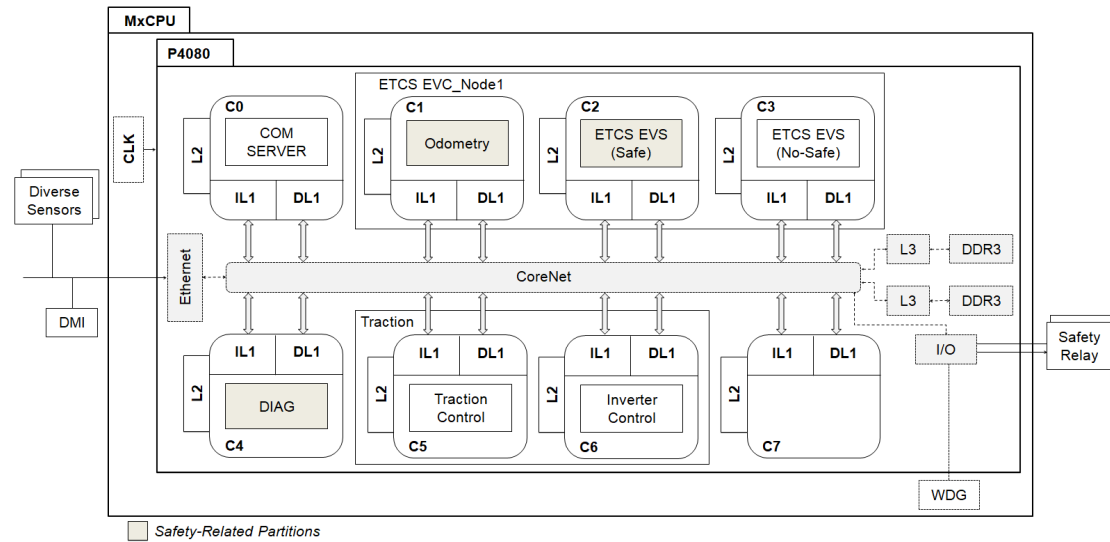
Figure 5.2: Safety concept (one processor; partitioned).

- * *P3-ETCS EVC (No-Safe)* performs additional non safety-related ETCS functionality (e.g., warning signal activation and information logging).
- The non safety-related traction computing nodes are partitioned according to their functionality:
 - * *P4-TRACTION CONTROL UNIT* executes non safety-related traction control.
 - * *P5-INVERTER CONTROL UNIT* executes non safety-related inverter control.
- The mixed-criticality node includes two additional partitions that inherit the maximum safety integrity of the partitions present in the system (*SIL* 4):
 - * *P6-COM SERVER* manages communication with other EVC nodes and peripheral subsystems.
 - * *P7-DIAG* to implement safety-related diagnostic measures.
- Hypervisor: Partitions are implemented on top of a certifiable hypervisor that provides the required virtual environment in a safe, transparent and efficient way.
- External *WDG*: The *MxCPU* integrates a separate *WDG* for diagnosis.

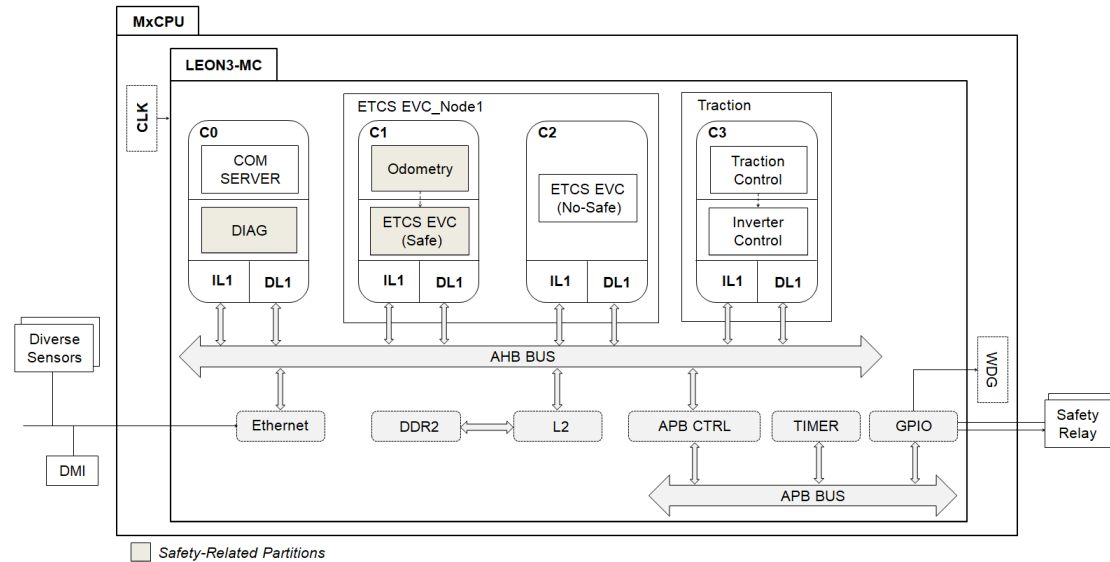
5.2 Railway Safety Concept

Multicore Safety Architecture

The partitioned integrated approach is evaluated on top of two particular multicore processors, the COTS NXP's P4080 (P4080) and ad-hoc LEON 3 based multicore implementation (LEON3-MC) (COB16). Figure 5.3 depicts the partition allocation to the P4080 and LEON3-MC multicores and shows their main architectural features of relevance:



(a) *MCN_SECU* detailed deployment in P4080 COTS processor



(b) *MCN_SECU* detailed deployment in ad-hoc LEON3-MC

Figure 5.3: Safety concept in partitioned multicore processor.

- **COTS P4080 Processor.** As the P4080 comprises 8 cores and the railway application is implemented by 7 partitions, each partition is allocated to one core as depicted in Figure 5.3a. The empty core could be used for additional diagnostics or to integrate additional applications.
- **Ad-hoc LEON3-MC.** Similarly, Figure 5.3b illustrates the ad-hoc safety concept on the LEON3-MC 4-core implementation. In this case, some of the cores integrate two partitions.

5.2.4 Failure Analysis

The fault hypothesis for the *MxCPU* is defined first, followed by the FMEA for partitions of the railway case study.

Fault Hypothesis

Based on the generic fault hypothesis defined in Section 4.4.2, the fault hypothesis for the railway mixed-criticality multicore integration is as defined below:

Federated to Integrated - Assumptions. The integrated multicore solution requires further safety measures with respect to the federated approach. Although the same safety attributes apply at system level (e.g., off-chip TMR redundancy), several new challenges arise at the mixed-criticality node level:

- Software safety partitions *P1* Odometry and *P2* ETCS EVC (safe) are provided as *SIL* 3 compliant items.
- Same safety life-cycle and FSM measures (Section 5.2.2-A) apply for system, platform and software applications.
- External TTEthernet/black channel communication is now replaced by the internal on-chip communication of the partitioned processor.
- Communication between the three redundant nodes remains the same (TTEthernet/black channel as defined in Section 5.2.2-D).
- Same digital input / output management (Section 5.2.2-E) and safe state requirements (Section 5.2.2-F) apply.
- Diagnostic techniques described in (Section 5.2.2-G) must be extended with particular techniques for the selected platform and to prevent from the hazards that the sharing of a single computing platform involves.

5.2 Railway Safety Concept

Unit of Failure. The impact of faults in the MxCPU is delimited by the following fault containment regions:

- The *processor* (*P4080* or *LEON-3 MC*) forms a single physical *FCR*.
- The *external WDG* forms another physical *FCR*.
- Each *partition* forms a single design *FCR*.

Failure Modes and Frequency of Failures. The *MxCPU* is provided as a *SIL 3* IEC 61508 compliant item, also compliant with EN 5012X, composed of platform and hypervisor ported to this platform. The *MxCPU* can fail in an arbitrary failure mode with the permanent failure rate in the order of 10-100 *FIT* and the transient failure rate in the order of 100,000 *FIT*:

- The processor can fail in an arbitrary failure mode.
- A partition can fail in an arbitrary failure mode when it is affected by a fault, both in the temporal as well as in the spatial domain (as defined in Table 4.1).
- The hypervisor (ported to the P4080 and LEON3-MC) provides independence of execution as described in G5 (Figure 4.8) (spatial independence and bounded temporal interference) among partitions. It can fail in an arbitrary failure mode when it is affected by a fault.

Failure Mode and Effect Analysis

The generic failure modes for partitions of Table 4.1 described in Section 4.4.2 apply to each of the safety partitions of the railway application, i.e., P1-ODOMETRY, P2-ETCS EVC (Safe), P6-COM SERVER, and P7-DIAG. Table 5.4 extends this FMEA with the most relevant hardware resources that may cause interferences in the selected multicore architectures¹:

P4080:

- Potential spatial interference sources in the platform are the shared SRAM and addressable I/O devices and may be originated in any platform component with bus master access rights (cores, interrupts, DMA, I/Os).

¹Further interference sources may exist like other low level hardware features, software execution path or initial conditions.

- Temporal interferences may happen in the event of simultaneous accesses to the shared resources present in the P4080 (L2 and *CPC* shared caches, main memory, CoreNet interconnect, shared peripherals). In addition, some P4080 components introduce potential execution time variability and temporal undeterminism that shall be considered in the design to avoid potential temporal interferences. These features include the memory coherency, impact of caches and barely documented features like the CoreNet.

LEON3-MC:

- Spatial interferences in the LEON3-MC may occur in the shared SRAM memory and addressable I/O devices.
- The main platform component amenable to temporal interferences in the form of multicore contention is the system AHB bus, as it is the responsible of arbitrating parallel requests. In addition, as in the case of the P4080, some platform components may lead to execution time variability like cacheable memory accesses (in private L1 and shared L2 cache) and the impact of the coherency protocol.
- As a result of the integration of partitions in the same core, interferences may also originate in core local resources of the LEON3-MC (processor time, L1 cache line evictions, TLBs, etc.).

5.2.5 Safety Measures

This section describes the safety measures that support the railway safety concept in compliance to the safety arguments defined in Section 4.4. Table 5.2 provides a summary of the measures and shows IEC 61508 equivalence with respect to railway EN 5012X standards.

G.4. Measures for the avoidance of systematic faults: As defined in the federated safety concept (Section 5.2.2), the system and partition development shall comply with *SIL* 4 FSM and all software safety partitions are *SIL* 3 compliant. In addition, the mixed-criticality approach requires that:

G4.1. <FSM>, *P6*-COM SERVER and *P7*-DIAG partitions are developed in compliance to *SIL* 3 IEC 61508 / *SIL* 4 EN 50128.

5.2 Railway Safety Concept

Table 5.2: Summary of safety measures and IEC 61508 to EN 5012X references.

Safety Measure (Goal)	Implementation	Reference to Standards	
		IEC 61508 (Strategy)	EN 5012X
G4 Avoidance of systematic faults in the safety life-cycle (Figure 4.7)			
<FSM> (G4.1)	SIL 4 compliant	Part 1 clause 6 (S4.1).	EN 50126 clause 5.
<Fault prevention> (G4.2)	Measures for P6 COM SERVER, P7 DIAG, SW/HW integration.	Part 2 Annex B (HW) / Part 3 Annex A (SW) (S4.2).	EN 50129 Annex E (HW) / EN 50128 Annex A (SW).
<Compliant item> (G4.3)	P1, P2, P6, MxCPU and hypervisor SIL3 compliant items.	Part 2 and Part 3 Annex D (S4.3).	EN 50126 clause p. 4.7.6.c.
G5 Safe co-existence of mixed-criticality applications (Figure 4.8)			
<Spatial Independence> (G5.1.1)	Hypervisor-based virtualization.	Part 3 Annex F (S5.1).	EN 50128 clause 7.3.4.9.
<Temporal Independence> (M5.1.2)	Hypervisor-based scheduling supported by MBPTA.	Part 3 Annex F (S5.1).	EN 50128 clause 7.3.4.9.
<Inter-partition Communication> (G5.1.3)	Hypervisor-based communication.	Part 3 Annex F (S5.1).	EN 50128 clause 7.3.4.9.
<Safe Partitioning> (G5.1.4)	Certifiable hypervisor.	Part 3 Annex F (S5.1).	n/a
<Configuration> (G5.1.5)	Defined by system architect using qualified tools.	n/a	n/a
<Independence violations> (AG5.2)	Away goal to G6.2.	Part 3 p. 7.4.2.9 (S5).	n/a
G6 Control of random hardware, systematic, environmental and operational errors at runtime (Figure 4.9)			
<Diagnostic Mechanisms> (G6.1)	SIL 3, HFT = 0, DC ≥ 99% techniques in MxCPU.	Part 2 Annex A (S6.1).	EN 50129 Table E.5.
<Independence violations> (G6.2)	Detected by hypervisor, DIAG or WDG.	Part 3 p. 7.4.2.9 (S6.2).	n/a
<System Reaction to Errors> (G6.3)	Safe state by SF (relay), or diagnostics (reset).	Part 2 p. 7.4.8 (S6.3).	EN 50129 Annex B.3.4.

G4.2. <Fault prevention>,

- The development of new *P6*-COM SERVER and *P7*-DIAG software partitions adopts measures for the avoidance of systematic faults according to IEC 61508-3 Tables A.2-A.5 / EN 50128 Tables A.2-A.11 (e.g., Defensive Programming).
- The integration of software partitions and the hardware platform and system validation includes systematic fault prevention activities compliant with IEC 61508-3 Tables A.6-A.7 and EN 50128 Table A.5 (e.g., Performance testing) and takes into account all *safety manuals* for compliant items.

G4.3. <Compliant item>,

- The *MxCPU* is provided as a *SIL* 3 compliant item with associated *safety manual*.
- The hypervisor meets the requirements of a certifiable hypervisor (*SIL* 3 compliant item) and it is integrated in the system in accordance to its *safety manual*.
- Safety partitions *P1*, *P2*, *P6* are provided as *SIL* 3 compliant items with associated *safety manual*. They include the techniques adopted in the federated approach to achieve software systematic capability *SIL* 3 (EN 50128 *SIL* 4) of safety partitions. The system architect shall define the additional hardware-software integration and system safety validation procedures according to *safety manuals*.

G.5. Measures for the co-existence of mixed-criticality applications: Mixed-criticality co-existence in the railway case study shall be supported by the features provided by the *SIL* 3 compliant hypervisor. The integrated hypervisor shall provide the following minimum services according to the goal structure of Figure 4.8:

G5.1.1. <Spatial independence [Memory]>: The hypervisor assigns a protected (virtual) memory region to each partition and manages the hardware MMUs.

G5.1.1. <Spatial independence [I/O]>: Safety-related peripherals are exclusively managed by safety partitions:

- ‘ETCS EVC (Safe)’ partition controls associated digital outputs and inputs for safety-relays (command digital output and confirm state with digital input).
- ‘DIAG’ partition controls associated digital outputs and inputs for *WDG* (command digital output and confirm state with digital input).

5.2 Railway Safety Concept

- The ‘COMM SERVER’ partition manages the Ethernet communication bus.
- G5.1.2. <Temporal independence>:** The hypervisor implements a cyclic schedule of partitions and ensures that time slots are assigned as statically configured (LPA+15).
- The system architect defines a static scheduling table for all partitions supported by WCET analysis that guarantees bounded temporal interference (G7).
- G5.1.3. <Inter-partition communication>:** The hypervisor supports mechanisms that allow safe data exchange between:
- All safety partitions to DIAG: $[P1 \rightarrow P7]$, $[P2 \rightarrow P7]$, $[P6 \rightarrow P7]$
 - ‘COM SERVER’ to Odometry: $[P6 \rightarrow P1]$
 - ‘Odometry’ to ‘ETCS EVC (Safe)’: $[P1 \rightarrow P2]$
- G5.1.4. <Safe partitioning>:** The hypervisor is compliant with *SIL* 3 requirements and its execution is in privileged mode, isolated and protected against external software faults.
- G5.1.5. <Configuration>** is static and defined during design stage with qualified tools. The hypervisor must start up, configure and initialize in a known, repeatable and correct state within a bounded time (e.g., internal data structures, virtualized resource initialization, etc.).

In addition, the safety hypervisor includes a fault-tolerant time synchronization that provides a global notion of time to the hypervisor partition scheduler and *health monitoring* to control random and systematic failures at hypervisor or partition level.

G.6. Measures for runtime error control: The system integrates the features to detect and control errors at runtime according to what it is specified in the argument structure of Figure 4.9.

G6.1. <Diagnostic Mechanisms>: The *MxCPU* includes diagnostic measures equivalent to the federated architecture, achieving a high diagnostic coverage ($DC \geq 99\%$, *SIL* 3, *HFT* = 0). Table 5.5 gathers the diagnostic measures defined for the railway case study. Those mechanisms are classified according to goal G6.1 of Figure 4.9 as follows:

- Autonomous hardware diagnostics and software commanded diagnostics are implemented in all platform components that participate in the execution of the safety function in accordance to IEC 61508 Annex A / EN 50129 Table E.5.

- Platform independent diagnostics:
 - Safety-related digital outputs connected to the *WDG* and safety-relays confirm their state by associated digital inputs and assigned partitions ('DIAG' and 'ETCS EVC (Safe)') periodically write (and read back) a dynamic signal on the outputs.
 - Off-chip redundancy: Reciprocal comparison by software for three channels, 2oo3 input comparison/voting for safety-related sensors.
 - The hypervisor includes *health monitoring* for the detection of hypervisor or partitions level errors.
- *MxCPU* Diagnostics:
 - Independent *WDG* controlled by the 'DIAG' partition for the combination of timing and logical monitoring of program sequence.
 - Diverse dual-channel output for the shut down path for safe state activation (electric and pneumatic braking systems) with 2oo3 voting of output signals.

G6.2. <Independence violations> The hypervisor detects and handles spatial independence violations with hardware MMU support. For timing interferences the 'DIAG' partition implements software diagnostics by periodically monitoring the completion of all partitions and controlling the external *WDG* accordingly.

G6.3. <Error reaction> Safe state is achieved by the safety chain depicted in Figure 5.4, either by the safety function (that commands safety-relays to stop the train) or by the different diagnostics that act in the *WDG* and force a *MxCPU* reset in case of error detection. Table 5.6 defines the system reaction to main subsystem errors.

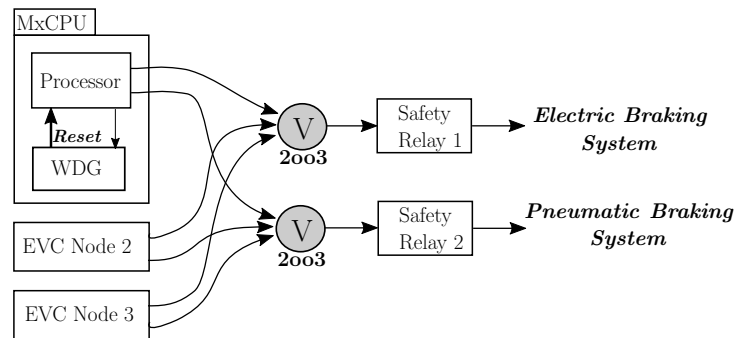


Figure 5.4: Safety chain for ETCS safe state activation.

Independence in Multicore Architectures

The use of a safety hypervisor reduces the burden of the transition from the integrated abstract approach to particular multicore implementations as the hypervisor is the responsible of providing independence guarantees among partitions as required in *G5.1*.

<**Spatial independence**> (*G5.1.1*) is ensured by hypervisor configuration:

- P4080: The hypervisor manages the memory MMU and I/O MMU (*PAMU*) available in the P4080 to avoid the presence of spatial interferences. The system architect is the responsible of assigning safety peripherals to exclusive safety partitions.
- LEON3-MC: Spatial interferences are managed by assigning different MMU-protected memory ranges to each partition in the hypervisor.

Similarly, the hypervisor provides non-blocking <**Inter-partition communication**> (*G5.1.3*) interfaces and protects the <**Configuration**> (*G5.1.5*) of critical settings against unintended modifications.

G7. <**Temporal independence**> in the railway multicore processors requires considering the specific platform resources (memories, peripherals, communication buses) and mitigating the effects of execution time variations and interferences that can lead to the partition level failures analyzed in the FMEA of Table 5.4. To this end, the system architect shall mitigate the effects of temporal interferences in the design through the configuration settings and a convenient timing analysis technique, in this case MBPTA:

G7.1. <**Platform configuration**>:

- P4080: Execution time variations caused by cache line evictions in the *CPC* shared caches are avoided by partitioning the *CPC* in 7 segments assigned to the different software partitions.
- LEON3-MC: L2 cache is partitioned across the 4 cores avoiding, in this way, the effects of shared cache line evictions by contender partitions. For those partitions sharing the same core, they are sequentially executed in a static cyclic scheduling basis implemented by the hypervisor and supported by WCET (MBPTA).

G7.2. <**Bounded temporal interference**> is ensured by means of MBPTA:

G7.2.1. <**Timing analysis**>: MBPTA is used to upper-bound the impact of execution time variation (e.g., caches) and multicore contention sources (i.e., simultaneous access requests) in the WCET

estimates computed for both the P4080 and LEON3-MC. To this end, MBPTA is applied to the three MBPTA-compliant platform designs introduced in Section 3.3: P4080-DSR, LEON3-MC-HR and LEON3-MC-DSR.

The pWCET resulting from the hardware randomization approach, is representative for any load in the contending cores, as the arbitration policy in the bus is modified to upper-bound the number of contenders and their bus access latency; and time randomized to determine which core is granted access to the shared cache. In this way, the LEON3-MC-HR ensures fully time composable estimates (i.e., valid for any contending software partition) without making any assumption on the contenders that will be executed in the other cores at operation time (Figure 5.5b).

With software randomization instead, bus contention needs to be managed offline. One solution to handle bus jitter is to deterministically upper bound it by monitoring the number of requests issued by the software partition under analysis (through Performance Monitoring Counters (PMC)s) and applying a contention model that upper-bounds the interference it can suffer in the presence of co-runner tasks (DFK+17). In the railway case study, as all software partitions are known a priori from the federated implementation, the actual requests of the contenders can also be monitored to estimate tighter partially time composable bounds (i.e., only valid for the analyzed contenders). Accordingly, DSR solution provides estimates for the particular railway contenders executed in two contending cores (Figure 5.5c).

As an illustrative example of the application of MBPTA in the railway case study the pWCET of ‘Emergency Brake’ safety function ($P2$) is obtained in the three platform setups described above. Figure 5.5 shows the pWCET curve resulting from the application of MBPTA to the longest execution time path of the railway application subset and Table 5.3 provides WCET values at relevant exceedance thresholds. The analysis-time distribution (ATD) is also reported in the form of Complementary Cumulative Distribution Function (CCDF), to show that the pWCET curve always upper-bounds the observed data.

Table 5.3: pWCET at relevant exceedance thresholds for railway case study.

Platform	HWM	10^{-3}	%	10^{-6}	%	10^{-9}	%	10^{-12}	%
LEON-HR	34464	36259	5.21	39130	13.54	42001	21.87	44872	30.20
LEON-DSR	33914	34094	0.53	34298	1.13	34462	1.62	34667	2.22
P4080-DSR	13322	13716	2.96	14639	9.89	15579	16.94	16501	23.86

5.2 Railway Safety Concept

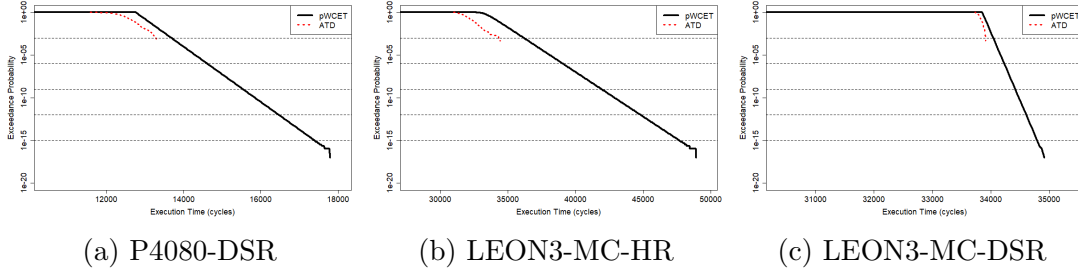


Figure 5.5: MBPTA application in the railway case study.

Results show tighter WCET results in the P4080 (Figure 5.5a) than in the LEON3-MC. This is explained by the increased performance of this architecture and the limited impact from multicore contention in the P4080 for the analyzed function, which fits in local L1 and L2 caches. Regarding the LEON3-MC architecture, DSR approach (Figure 5.5c) shows more accurate results than HR (Figure 5.5b) for the railway case study, as the contention impact of the particular railway contenders is used in the analysis. LEON3-MC-HR instead, provides increased flexibility and scalability as the computed bounds are also valid if the software applications in contending cores are updated or modified. In either case, results show that under the analyzed conditions ‘Emergency brake’ safety function’s WCET is below its timing deadline.

The pWCET projection allows to choose the safety margin based on exceedance probability with a formal justification of the chosen bounds, instead of using common-practice 20% margin based on experience or engineering practice. The resulting estimate provides independence of execution guarantees by the verification of the execution time budgets assigned to software tasks in the scheduling. Chapter 6 provides further details on the application of MBPTA, with special emphasis on its safety implications.

5.2.6 Detailed Tables

This section contains the detailed tables that the railway safety concept refers to.

Table 5.4: Simplified Failure Mode and Effect Analysis for railway case study (partitions).

Failure Mode	Failure Effect	Potential Cause	Multicore resource		Failure detection and compensation (MxCPU)
			P4080	LEON3-MC	
Not Executed	Safety function not performed.	Random HW fault			<Diagnostic mechanisms>.
		Design fault			<Fault prevention> to reduce the probability of systematic faults.
		Integration fault			
		Blocking execution by another partition	CoreNet, <i>CPC</i> , main memory, undocumented	Core, AHB/APB buses, L2 cache, main memory,	<Temporal independence> guaranteed by hypervisor, <Platform configuration>, supported by <Timing analysis>.
Incorrect (functional) Execution	Safety function performed with wrong data, incorrect outputs may be computed.	Memory / I/O corruption (random)			<Diagnostic mechanisms>.
		Memory corruption by contending partition	Main memory	Main memory	<Spatial independence> for memory by hypervisor managed MMU.
		Memory corruption by components with bus master rights	Interrupt controller, DMA	Interrupt controller, DMA	<Spatial independence> by hypervisor managed <i>PAMU</i> (P4080).
		I/O corruption by contending partition	Addressable devices	Addressable devices	<Spatial independence> for I/Os by hypervisor managed MMU.
Incorrect (temporal) Execution	Safety function does not compute its outputs on time and may cause dangerous deadline overruns.	Random HW fault			<Diagnostic mechanisms>
		Blocking execution by another partition	CoreNet, L2, <i>CPC</i> , main memory, cache coherency,	Core, local L1 cache, buses, L2 cache, main memory, cache coherency,	<Temporal independence> guaranteed by hypervisor <Platform configuration>, supported by <Timing analysis>.
		Incorrect allocation of execution time	FPU,		Impact of runtime SoJ captured in WCET

			undocumented	FPU	by means of MBPTA and HW/SW Rand.
		Blocking execution by inter-communication	CoreNet, main memory	AHB bus, main memory	<Inter-partition communication> by compliant item hypervisor.
		Integration fault			<Compliant item> integration tests.
'Write' access to non-assigned memory or I/O	Safety peripherals commanded by incorrect partition. Data written by safety partition may be invalidated.	Memory / I/O corruption (random)			<Diagnostic mechanisms>.
		Memory corruption	Main memory, addressable I/O, Hypervisor	Main memory, addressable I/O, Hypervisor	<Spatial independence> by hypervisor managed MMU.
		Faulty virtualization (wrong address translation)			<Safe partitioning>.
Unable to write in assigned memory or I/O	Safety partition outputs not consistent with expected value.	Memory / I/O corruption (random)			<Diagnostic mechanisms>.
		Faulty virtualization (wrong address translation)	Hypervisor	Hypervisor	<Safe partitioning>.
		Blocking shared resource	CoreNet, main memory, I/O	Core, AHB/APB buses, main memory	<Temporal independence> and exclusive I/O access by hypervisor, <Platform configuration>, supported by <Timing analysis>.
Multiple combinations of previous failure modes	Uncontrolled / unpredictable behavior of safety partition.	Faulty virtualization (privileged partition access)	Hypervisor	Hypervisor	<Safe partitioning>.
		Partition resets core and / or other core(s)			<Safe partitioning>. <Spatial independence>.
		Uncontrolled or not repeatable initialization, configuration, startup			<Configuration>.

Table 5.5: Railway system measures for runtime error control.

Subsystem	Technique	IEC 61508-7	Maximum DC	Description
Processing units	Reciprocal comparison by software for three channels	A.3.5	High	Three processing units (TMR) exchange data (including results, intermediate values and test data) reciprocally. A comparison of the data is carried out using SW in each core.
Invariable memory	Signature of a double word	A.4.4	High	Inclusion of 32 bit CRC in memory.
	Program code protection	n/a	n/a	Configuration parameters are not configurable in runtime. Test of program code memory and fixed data memory during boot procedure.
Variable memory	Test RAM	A.5.3	High	Periodic test “checkboard” or “march”.
Interconnect	Inspection using test patterns	A.7.4	High	SW commanded cyclic test with a defined test pattern to check the correctness of on-chip buses.
Clock	Watchdog with separate time base and window	A.9.2	High	Watchdog with upper and lower time windows and independent time base from the multicore clock.
Power supply	Voltage or current control	A.8.3	High	Power failure monitor that implements safety shut-off in case of power supply malfunction (under/overvoltage).
Program sequence	Combination of timing and logical monitoring	A.9.4	High	Watchdog with upper and lower time window, only refreshed if the result of monitoring user program execution is correct.
Interrupt controller	Majority voter	A.1.4	High	Comparison between the three processing units (TMR).
Digital I/O	Input comparison/voting	A.6.5	High	2oo3 voting among the three TMR processing units with and periodic dynamic test signal monitoring.
Actuators (relay, WDG)	Monitored outputs	A.6.4	High	External actuators monitored by means of a digital input to the MxCPU that represents the state of the contact.
Partition	Independence violation detection	n/a	n/a	Certifiable hypervisor that provides the minimum services defined in <i>G7.1/G7.2</i> of Section 5.2.5.
Platform	Built-in Self-Test (BIST) and start-up tests	n/a	n/a	Start-up tests during platform initialization. At operation further BIST shall be run periodically.

5.2 Railway Safety Concept

Table 5.6: Railway system reaction to errors.

ID	Error Group	System Reaction	Final State
SR1	Safety partition P1, P2, P6 error (as defined in Table 5.4).	<ol style="list-style-type: none"> 1. DIAG does not receive confirmation from faulty partition P1, P2 or P6. 2. DIAG does not refresh WDG. 3. MxCPU reset. 4. MxCPU reset de-energizes safety-relay. 	Safe state
SR2	P7 DIAG partition error (as defined in Table 5.4).	<ol style="list-style-type: none"> 1. DIAG does not refresh WDG. 2. MxCPU reset. 3. MxCPU reset de-energizes safety-relay. 	Safe state
SR3	Non-safety partition [P3 – P5] error (as defined in Table 5.4).	<ol style="list-style-type: none"> 1. DIAG does not receive confirmation from faulty partition. 2. DIAG re-starts the faulty partition. 	Safety partitions remain operational
SR4	P4080/LEON3-MC platform error detected by diagnostics.	<ol style="list-style-type: none"> 1. Error detected by diagnostics. 2. DIAG does not refresh WDG. 3. MxCPU reset. 4. MxCPU reset de-energizes safety-relay. 	Safe state
SR5	P4080/LEON3-MC platform error undetected by platform diagnostics.	<ol style="list-style-type: none"> 1. WDG is not refreshed. 2. WDG reaches time-out and resets MxCPU. 3. MxCPU reset de-energizes safety-relay. 	Safe state
SR6	Error in external WDG.	<ol style="list-style-type: none"> 1. Safety function remains operational. 2. Error detected by DIAG partition. 3. MxCPU reset. 4. MxCPU reset de-energizes safety-relay. 	Operational / Safe state
SR7	Random error in safety-relay.	<ol style="list-style-type: none"> 1. Error detected by DIAG partition (periodic test by dynamic signal generation and read back). 2. DIAG does not refresh WDG. 3. MxCPU reset. 4. MxCPU reset de-energizes safety-relay. 	Safe state

5.3 Automotive Safety Concept

This section presents the safety concept definition for the automotive case study. The case study comprised of the subsystems introduced in Section 3.2.2: a safety-critical Cruise Control (CC) system and a non safety-related power window controller. In this case, as opposed to the railway case study of Section 5.2, partitioning is enforced by hardware and software enabled techniques with the aim of reducing the overhead introduced by the hypervisor. As in the case of railway, the safety concept is derived from the IEC 61508 safety argumentation presented in previous sections. As a result, the safety concept definition includes many references to IEC 61508 and specific allusions to automotive ISO 26262 standard are made where required.

5.3.1 Safety Requirements

The CC shall perform a set of *safety goals* compliant with ISO 26262 *ASIL D* requirements. In this case study, the following *safety goal* and associated safety requirements for the CC system (SR_CC in Table 5.7) are defined as an example for the definition of the safety concept:

- SG-1: Avoid the inability to deactivate CC when required (*SIL 3 / ASIL D*).

Table 5.7: Main safety requirements of the cruise control system.

ID	Requirement
SR_CC_1.A	The safety goal “Cruise Control Deactivation” avoids the inability to deactivate the CC when required. In case of a fault leading to inability to deactivate CC, the engine control unit shall switch to a ‘safe state’ within the <i>PST</i> / <i>Fault Tolerant Time Interval (FTTI)</i> .
SR_CC_1.B	“Cruise Control Deactivation” must be provided with <i>SIL 3 / ASIL D</i> level.
SR_CC_1.1	The ‘CC commands’ transmitted by the Cruise Control Signal Acquisition functional unit shall be consistent with the status of the buttons (set, speed+, speed-, off, resume).
SR_CC_1.2	The “Cruise Control Monitor” function shall generate a ‘Cruise Control Disengagement’ signal consistent with the input button/pedal requests.
SR_CC_1.3	The ‘safe state’ shall be achieved by deactivation of the Cruise Control System (by commanding safety digital-outputs connected to external safety-relays).
SR_CC_1.4	The <i>PST</i> (IEC 61508) / <i>FTTI</i> (ISO 26262) is 1 second.

5.3 Automotive Safety Concept

5.3.2 Federated Safety Concept

The high level *SIL* 3 / *ASIL* D requirement SR_CC_1_A is assigned to the CC signal acquisition and CC monitor and control functions allocated in two different ECUs in a federated paradigm and separated from other subsystems as depicted in Figure 5.6.

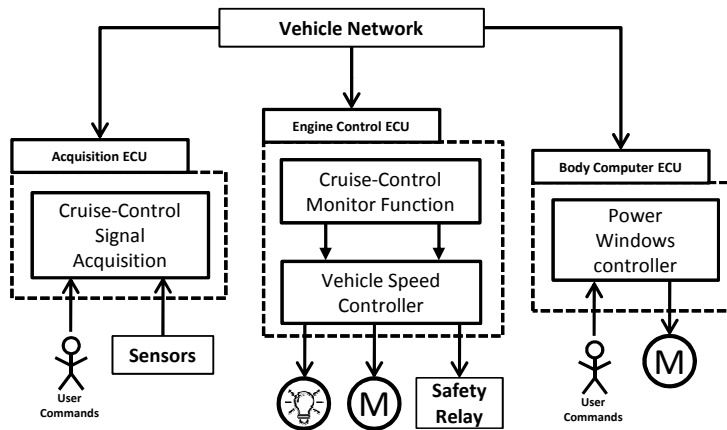


Figure 5.6: Federated system architecture for the automotive use case.

The safety-critical acquisition and engine control ECUs execute safety software compliant with the following features:

- A. *Life-cycle and development tools*: The system, platform and software development of the CC system is compliant with safety life-cycle (ISO 26262 Figure 1 / IEC 61508-1 clause 7) and corresponding *FSM* (ISO 26262-2 / IEC 61508-1 clause 6).
 - A.1. The CC system is developed using *qualified tool* and compilers (*ASIL* D / *SIL* 3).
 - A.2. The system architect takes reasonable measures to develop a safe product taking into consideration the *safety manuals* of all *compliant items* / *Safety Element out of Context (SEooC)* (ISO 26262-10 clause 4 / IEC 61508 parts 2 and 3 Annex D).
- B. *Architecture*: The stringent size-weight-power requirements of the automotive domain result in prevailing single channel *1oo1* safety architectures. Despite redundancy is considered at component level (e.g., dual-core lockstep

setups, redundant communication buses) completely redundant hardware architectures (e.g., *TMR*) are out of the scope of ISO 26262 standard. Accordingly, in this concept definition each ECU adopts a *1oo1* single channel architecture:

- B.1. Platform *compliant item/SEooC: Signal Acquisition* and *Engine Control* ECUs, composed of hardware and associated platform software, are ISO 26262 *ASIL-D SEooC* / IEC 61508 *SIL 3 compliant items* (see A.2).
- B.2. Each ECU executes a functionally independent software application (see C).
- B.3. The vehicle communication bus is shared with other ECUs (see D).
- B.4. Each ECU is physically separated from other ECUs and has independent resources (i.e., memories, peripherals, clock).
- C. *Software Application*:
 - C.1. *ASIL D* (IEC 61508 *SIL 3*) life-cycle and *FSM* compliant software development for *Signal Acquisition* and *Engine Control* ECUs.
 - C.2. Safety-related inputs, subsystems and outputs are managed exclusively by associated safety-related software tasks.
 - C.3. After start-up the safety software application commands digital-outputs according to SG-1 and diagnosis status (a failure corresponds to digital-output deactivation).
- D. *Vehicle communication bus*: Safe communication protocol that provides safety techniques and attributes of interest:
 - D.1. CAN based Time-Triggered communication protocol (TTCAN):
 - D.1.1. Compliant with ISO-11898-4 ([ISO-11898](#)) specification (e.g., minimum bus length, signaling rate, maximum number of nodes, cabling, etc.).
 - D.1.2. It establishes a global notion of time by means of a Time Master (TM) that transmits a synchronization message to all the nodes (ISO-11898-4).
 - D.1.3. It provides freedom from interference among safety-related and non safety-related communication.
 - D.2. Analogous black channel safety communication that provides equivalent safety properties.
- E. *External interfaces*:

5.3 Automotive Safety Concept

- E.1. The *Signal Acquisition* ECU reads safety-related digital inputs for the CC driver interface buttons (SET/SPEED+, SPEED-, OFF, RESUME) and safety-related analogue inputs (break, accelerator and clutch pedals and speed sensor).
 - E.2. The *Engine Control* ECU exclusively manages an external safety-relay through a digital output for the hardware deactivation of the CC (safe state activation).
 - E.3. The fail safe state of the safety digital output is de-energized (see F).
 - E.4. The default state of the safety digital output during no-power and initialization is de-energized (inherent fail safe state).
 - E.5. The safety-relay is monitored by means of a digital input to the *Engine Control* ECU that represents the state of the contact.
- F. *Safe State*:
- F.1. Safe state is defined as the de-activation of the CC system (giving back control to the driver) as defined in the safety requirement SR_CC_1.3 of Table 5.7.
 - F.2. Whenever there is a violation of a *safety goal* or if the system, hardware or software diagnosis detect an error, the safe state must be reached within the *FTTI* / *PST* and maintained.
- G. *Diagnosis*: The system includes diagnostic measures with a high coverage (DC >99%) for ASIL D (ISO 26262-5 Annex D).
- G.1. Diagnosis with a high coverage are implemented on all relevant components of each safety-related ECU (e.g., signature of a double word in memories, Power Failure Monitor, watchdog timer, clock monitoring, I/Os, communication, etc.) (ISO 26262-5 Tables D.2-D.14 and IEC 61508-2 Tables A.2-A.14).
 - G.2. The safety software application implements additional life-cycle related techniques (e.g., ISO 26262-6 Table 8 / IEC 61508-3 Table A.4 defensive programming), error correction/detection techniques and complements the diagnosis techniques implemented in the platform (e.g., refresh watchdog, monitor external safety-relays).
 - G.3. The system architect implements additional diagnosis and verification measures taking into consideration the *safety manuals* of all *compliant items* (ISO 26262-10.4 / IEC 61508 part 2 and 3 Annex D).

5.3.3 Integrated Node Architecture

The integrated solution hosts the federated applications of the automotive case study in a single partitioned processor.

Integrated Safety Architecture

The overall safety architecture is depicted in Figure 5.7 and described below:

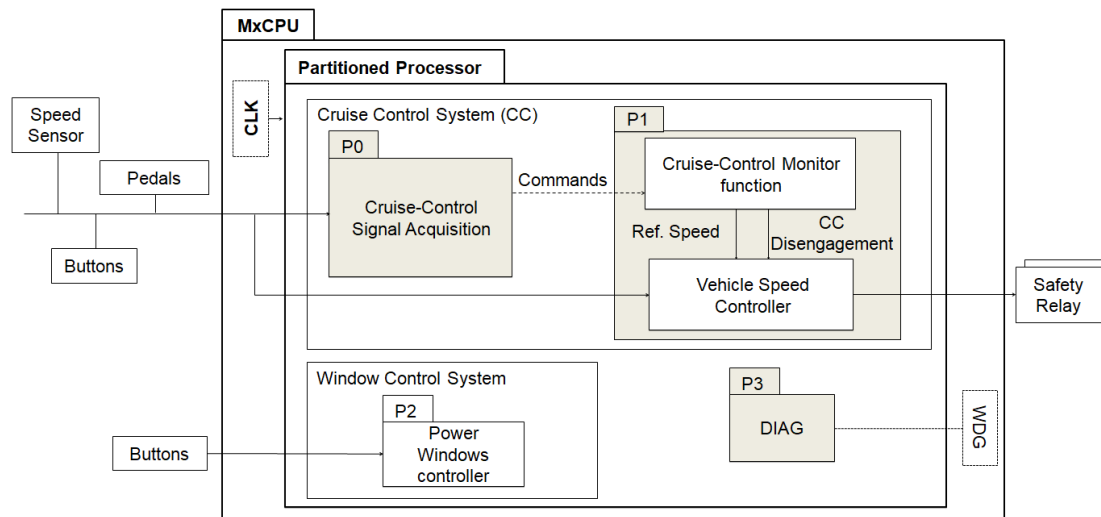


Figure 5.7: Automotive safety concept (one processor, partitioned).

- *MxCPU*: Composed of partitioned processor with hardware partitioning support and external *WDG*.
- Partitions: Equivalent to the ECU distribution of the federated system and additional partition for diagnostics:
 - *P0* encapsulates the *CC Signal Acquisition* ECU.
 - *P1* includes both the *CC* monitor and control functionality which are executed one after the other in a sequential manner.
 - *P2* hosts the non safety-critical power windows controller.
 - *P3* *DIAG* performs software-based diagnostics, configuration and protection activities.
- Processor: Partitions are enforced by hardware protection mechanisms. These means shall prevent non-safety partitions from interfering with safety partitions and freedom from interference among safety partitions.

5.3 Automotive Safety Concept

- External *WDG*: Separate watchdog timer for diagnosis and error reaction.

Multicore Safety Architecture

The case study is evaluated on the state-of-the-art AURIX processor family ([AURIX](#)) that integrates three cores as introduced in Section 3.1.3. Figure 5.8 illustrates the partition to core allocation and the main hardware resources of relevance of the AURIX TC27X architecture. Safety-related partitions $P0$ - $P1$ are allocated to cores $c0$ and $c1$ as they provide increased safety by means of lockstep processor redundancy. $P3$ has also safety implications and it is therefore executed in core 0 together with partition $P0$. The non safety-related partition is allocated to the remaining core without redundancy.

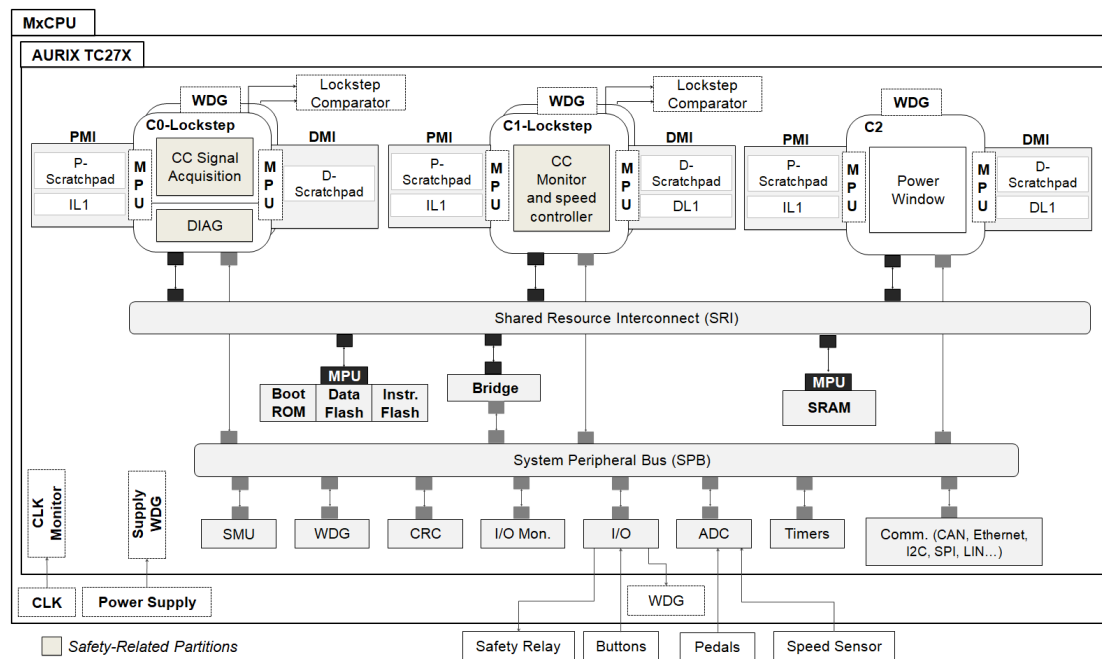


Figure 5.8: Safety concept in AURIX TC27X multicore processor

5.3.4 Failure Analysis

As part of the failure analysis process, the fault hypothesis and a simplified FMEA for the automotive case study are described below.

Fault Hypothesis

The fault hypothesis for the automotive mixed-criticality multicore integration is based on the generic fault hypothesis defined in Section 4.4.2:

Federated to Integrated - Assumptions. The following safety properties, described in the federated approach of Section 5.3.2, are assumed to be maintained when integrating the applications in a single platform:

- Software safety partitions $P0$ and $P1$ of the cruise control system are provided as *SIL 3 / ASIL D compliant items*.
- Same safety life-cycle and *FSM* measures described in Section 5.3.2-A apply for system, platform and software applications.
- External (off-chip) communication is replaced by on-chip communication.
- External interfaces are equivalent to those described in the federated implementation (Section 5.3.2-E) and the fail safe system requirements are maintained to reach the safe state (e.g., de-energization of safety-relay as described in Section 5.3.2-F).
- The definition of required diagnostic techniques described in Section 5.3.2-G can be reused taking into account the properties of the new hardware platform and additional diagnostics shall be implemented to prevent from the new hazards that the sharing of a single computing platform involves.

Unit of Failure. The impact of faults in the MxCPU is delimited by the following fault containment regions:

- The AURIX processor forms a single physical *FCR*.
- The external *WDG* forms another physical *FCR*.
- Each partition forms a single design *FCR*.

Failure Modes and Frequency of Failures. The AURIX platform is provided as an *ASIL D SEooC*. The *MxCPU* can fail in an arbitrary failure mode with the permanent failure rate in the order of 10-100 *FIT* and the transient failure rate in the order of 100,000 *FIT*:

- The AURIX can fail in an arbitrary failure mode. The residual failure rate of the AURIX with respect to hardware random failures is equal or lower than 0.1 *FIT* after applying the provided safety mechanisms as established in the *safety manual* (Inf14).
- A partition can fail in an arbitrary failure mode when it is affected by a fault, both in the temporal as well as in the spatial domain (as defined in Table 5.10).

5.3 Automotive Safety Concept

Failure Mode and Effect Analysis

The simplified FMEA for partitions of Table 4.1 shall be considered for each of the safety partitions of the automotive application: P0-Signal Acquisition, P1-CC Monitor and Control and P3-DIAG. This FMEA is updated in Table 5.10 with the main spatial and temporal interference sources of the AURIX processor that may result in partition failures:

- Spatial interferences among different cores may happen in the shared SRAM memory or addressable I/O devices.
- Regarding temporal interference sources, the effects of the arbitration policy shall be considered in the event of simultaneous accesses to shared resources in order to decide which core to serve first. While the main interconnect (SRI) implements a crossbar architecture, which allows several parallel requests, when the requests target the same resource (e.g., SRAM memory), they can affect each others timing. The peripheral bus (SPB) instead, is implemented as a shared bus, and provides mutual exclusive access to the connected peripherals. This causes significant impact on the timing of the different cores that try to access it at the same time. In addition to the impact of simultaneous accesses as explained above, all cacheable memory accesses can result in a hit or miss in cache, which creates another source of timing variability.
- As partitions *P0* and *P3* of the automotive system are allocated to the same core, core-local resources are also potential interference sources, such as, local scratchpad memories, L1 caches and the core itself.

5.3.5 Safety Measures

This section describes the safety measures to fulfill with the requirements of the safety argumentation of Section 4.4. Table 5.8 summarizes the safety measures that support the automotive safety concept and shows the ISO 26262 equivalence for the IEC 61508 based strategies defined in the goal structures of Section 4.4.

G.4. Measures for the avoidance of systematic faults: As defined in the federated safety concept (Section 5.3.2) the system and partition development is compliant with *SIL 3 / ASIL D FSM* and software safety partitions are *SIL 3 / ASIL D compliant items* with associated *safety manuals*. In addition, the mixed-criticality approach requires that:

Table 5.8: Summary of safety measures and IEC 61508 to ISO 26262 references.

Safety Measure (Goal)	Implementation	Reference to Standards	
		IEC 61508 (Strategy)	ISO 26262
G4 Avoidance of systematic faults in the safety life-cycle (Figure 4.7)			
<FSM> (G4.1)	SIL 3 compliant	Part 1 clause 6 (S4.1).	Part 2 (ASIL D).
<Fault prevention> (G4.2)	Measures for P4 DIAG and SW/HW integration.	Part 2 Annex B (HW) and Part 3 Annex A (SW) (S4.2).	Part 5 Annex D (HW) and Part 6 sections 7 to 9.
<Compliant item> (G4.3)	[P0 - P2] and MSCPU SIL3 compliant items.	Part 2 and Part 3 Annex D (S4.3).	Part 10 clause 4 ASIL D SEooC.
G5 Safe co-existence of mixed-criticality applications (Figure 4.8)			
<Spatial Independence> (G5.1.1)	HW partitioning in the AURIX.	Part 3 Annex F (S5.1).	Part 6 Annex D
<Temporal Independence> (M5.1.2)	AURIX configuration and WCET analysis (MBPTA).	Part 3 Annex F (S5.1).	Part 6 Annex D
<Inter-partition Communication> (G5.1.3)	Unidirectional channels based on protected shared memory.	Part 3 Annex F (S5.1).	Part 6 Annex D
<Safe Partitioning> (G5.1.4)	AURIX is ASIL-D compliant item.	Part 3 Annex F (S5.1).	Part 6 Annex D
<Configuration> (G5.1.5)	Defined by system architect using qualified tools.	n/a	n/a
<Independence violations> (AG5.2)	Away goal to G6.2.	Part 3 p. 7.4.2.9 (S5).	Part 6 p. 7.4.11
G6 Control of random hardware, systematic, environmental and operational errors at runtime (Figure 4.9)			
<Diagnostic Mechanisms> (G6.1)	SIL 3, HFT = 0, DC ≥ 99% techniques.	Part 2 Annex A (S6.1).	Part 5 Annex D
<Independence violations> (G6.2)	Detected by AURIX safety features, DIAG or WDG.	Part 3 p. 7.4.2.9 (S6.2).	Part 6 p. 7.4.11
<System Reaction to Errors> (G6.3)	Safe state by SF (relay), or diagnostics (reset).	Part 2 p. 7.4.8 (S6.3).	Part 5 p. 7.4.3.3

5.3 Automotive Safety Concept

G4.1. <FSM>, safety partition *P3* DIAG is conceived, developed and certified using a *SIL 3 / ASIL D FSM* (highest integrity level among all partitions in the system).

G4.2. <Fault prevention>,

- The development of new software partitions (i.e., *P3* DIAG) adopts measures for the avoidance of systematic faults according to IEC 61508-3 Tables A.2-A.5 / ISO 26262-6 sections 7-9.
- The integration of software partitions and the hardware platform and system validation includes systematic fault prevention activities compliant with IEC 61508-3 Tables A.6-A.7 / ISO 26262-5/6 section 10 (e.g., resource usage test) and taking into account all *safety manuals* for *compliant items*.

G4.3. <Compliant item>,

- The *MxCPU* is provided as an *ASIL D / SIL 3 SEooC* with associated *safety manual*.
- Safety partitions [*P0 – P1*] are provided as an *ASIL D / SIL 3* compliant items. Functional safety requirements of each item are contrasted with the functional safety requirements assumed for the *SEooC* (assumptions of use). The system architect shall define the additional hardware-software integration and system safety validation procedures according to *safety manuals*.

G5. Measures for the co-existence of mixed-criticality applications: Partitions in the automotive case study are enforced based on hardware support, by the configuration of protection mechanisms available in the *ASIL-D* compliant AURIX platform and an OS layer that provides minimal services. In this context, a partition is defined as an independent execution environment enforced by hardware and each partition shall comprise the following minimum features:

G5.1.1. <Spatial independence [Memory]>: The local memory of core 0 is segregated in at least two independent regions for the two partitions allocated to it. Similarly, shared memory is divided in different address ranges for each partition. Memory regions need to be protected against unintended accesses.

G5.1.1. <Spatial independence [I/O]>: Partitions are granted exclusive access to associated peripherals:

- CC monitor and control partition (*P1*) is granted exclusive access to the digital output connected to the external safety-relay.

- Diagnostics partition $P3$ has exclusive access to the digital output for controlling the external watchdog.

G5.1.2. <Temporal independence>: Platform configuration and a statically defined schedule of partitions implemented by the OS and supported by WCET analysis to ensure that time deadlines are preserved.

G5.1.3. <Inter-partition communication>: Safe communication is guaranteed by the system configuration not to jeopardize spatial and temporal independence.

G5.1.4. <Safe partitioning>: AURIX platform and OS are *ASIL D* compliant.

G5.1.5. <Configuration> is static and defined during design stage with *qualified tools*. The AURIX processor provides protection for safety configuration registers.

G.6. Measures for runtime error control: The system integrates the features to detect and control errors at runtime according to what it is specified in the argument structure of Figure 4.9.

G6.1. <Diagnostic mechanisms>: The system includes diagnostic measures to achieve a coverage equivalent to the federated architecture ($DC \geq 99\%$, $SIL\ 3$, $HFT = 0$). Table 5.11 gathers the diagnostic measures for relevant system components based on the following features:

- Autonomous hardware diagnostics: The AURIX architecture integrates diagnostic mechanisms in hardware (*‘AURIX Safety Features’*):
 - Cores: c0 and c1 are redundant (lockstep configuration). Each core has code and data MPUs for software tasks (16 data protection ranges, 8 code protection ranges) and a watchdog timer. All cores have independent reset and power management.
 - Shared memory: SRAM with configurable write protected regions (up to eight address ranges with a Tag ID for granting access rights). The memory system includes a Memory Test Unit (MTU) with features such as, configurable memory Built-in-Self-Tests (MBIST), error detection/correction via ECC, ROM content verification (initialization and data integrity checking) and multi-bit error detection.
 - On-chip buses (SRI and SPB): SRI with end-to-end monitoring of data and address failures and MPU in every SRI slave. Register access protection in SRI and SPB.

5.3 Automotive Safety Concept

- Additional measures:
 - * I/O monitoring (e.g., comparison with reference value) and redundancy (e.g., for Analogue to Digital Converters (ADC)).
 - * Two clock sources: crystal oscillator and backup clock with monitoring functions (over/under-frequency control).
 - * Power supply with power monitoring functions (over/under-voltage control).
- Software-commanded diagnostics: partition (*P3 DIAG*) manages and implements the required additional safety techniques:
 - Configures the ‘*AURIX Safety Features*’ at start-up (memories, internal watchdog timers, ADC, BISTs).
 - Performs the required additional periodic checks (I/O checks, communication bus, *WDG*) (IEC 61508-7 A.6.1 / ISO 26262-5 D.2.6.1 test pattern).
 - Controls the external watchdog (*WDG*).
- Platform independent diagnostics:
 - Safety-related digital outputs connected to the *WDG* and safety-relay are monitored by associated digital inputs (IEC 61508-7 A.1.1 / ISO 26262-5 D.2.1.1: failure detection by online monitoring).
- *MxCPU* diagnostics:
 - Independent *WDG* controlled by *P3 DIAG* partition for the combination of timing and logical monitoring of program sequence (IEC 61508-7 A.9.4 / ISO 26262-5 D.2.9.4).

G6.2. <Independence violations> The hardware mechanisms protect memory space and I/Os against independence violations. For timing interferences, the correct timing behavior is monitored by the core local watchdogs and *P3 DIAG* partition implementing software diagnostics that periodically monitor the completion of all partitions.

G6.3. <Error reaction> Safe state is achieved either by safety function (*P1* commanding safety-relay) or diagnostics that act in the *WDG* and force a *MxCPU* reset in case of error detection (Figure 5.9). Table 5.12 defines the system reaction to main subsystem errors.

Independence in Multicore Architectures

The implementation details to achieve the aforementioned independence guarantees in the AURIX multicore are described below.

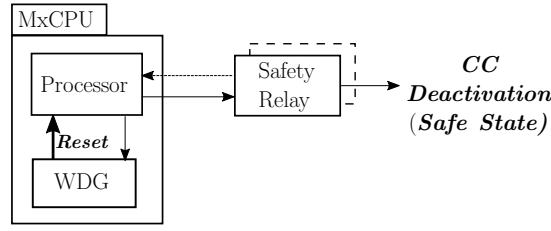


Figure 5.9: Safety chain for CC safe state activation.

<Spatial independence> (G5.1.1) is achieved by the segregation and protection of memory areas for each individual software partition:

- Local memories: Local memories on each core of the AURIX can be configured in four memory *Address Protection Sets (APS)* per CPU, with specific permissions for code and data accesses based on the address generated by an application. Figure 5.10a shows the configuration of the different memory regions for the partitions executing on core 0. In each time slot of core 0 only one of the address protection sets is active (Figure 5.10b). In this way, in each partition switch or if an interrupt or trap handler is entered, the associated address protection set is selected and accesses from other partitions are not permitted. The OS kernel is the responsible for setting up the address protection correctly on each context switch. This OS service shall be developed according to *ASIL D* safety requirements.

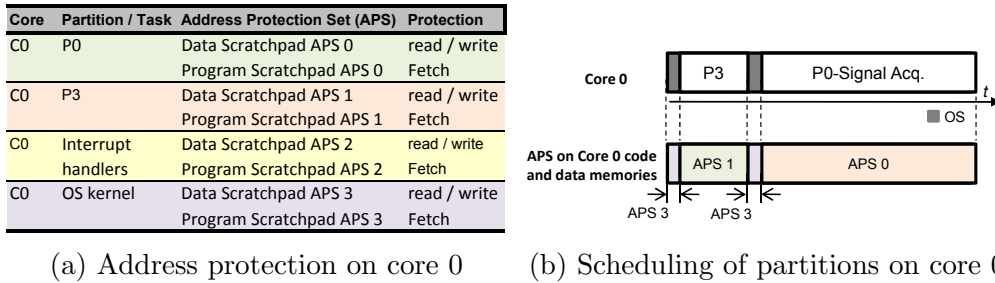


Figure 5.10: Memory and time segregation on core 0.

- Shared memory: The shared SRAM memory can be configured in eight write protected address ranges. A *Protected Memory Region (PMR)* is configured for each safety-critical partition as shown in Figure 5.11 (*PMR* 0 for *P0*, *PMR* 3 for *P3* and *PMR* 4 for *P1*). In each *PMR*, write accesses are protected by checking the ID used to initiate SRI master transactions (master TAG ID). Each core has two master interfaces (code and data) to access to the SRI, each of it with a unique ID. For core 0, where two different partitions are executed, the platform gives the option to configure two additional

5.3 Automotive Safety Concept

IDs on the same core, by identifying the partitions as safe (C0_DATA_S) or regular (C0_DATA_NS).

- I/Os: Addressable on-chip resources are protected by a firewall mechanism where the access to the SRI and SPB slaves is only granted to the masters configured with such rights. Accordingly, partitions P1 and P3 are configured with exclusive access rights to independent GPIO port lines (Figure 5.11b).

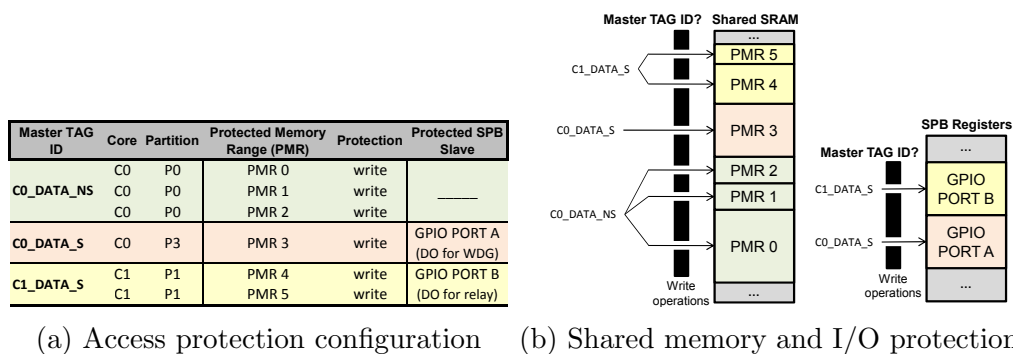


Figure 5.11: Platform protection mechanisms against spatial interferences.

Safe <**Inter-partition communication**> (G5.1.3) is guaranteed by means of shared SRAM memory. Each shared memory is configured as a write protected address range where only one partition has write permissions. Accordingly, inter-partition communication channels are unidirectional with an independent shared memory for each of the required communication channels. Three additional protected memory regions are defined in Figure 5.11 for the communication among *P0* and *P1* (PMR 1), *P0* and *P2* (PMR 2) and *P1* and *P2* (PMR 5).

The <**Configuration**> (G5.1.5) of critical settings such as, partition to core allocation, protection sets, routing of interrupts to cores, scheduling and arbitration policies in the SRI and SPB, watchdog time windows, etc.. is protected. On the one hand, configuration registers can only be accessed in supervisor mode and a register protection mechanism is also in place to define which masters may have modification rights on the safety-related registers (in this case only *P3* DIAG). On the other hand, after system initialization, whenever the software in supervisor mode and from an authorized master attempts to modify a safety configuration register, it first needs to unlock a password protected bit. Once this bit is unlocked, the software is allowed to modify the registers within a predefined time period protected by the watchdog (the password protected bit needs to be locked again before watchdog time-out).

G7. <Temporal independence> is achieved by mitigating possible time interferences among partitions (as those analyzed in the FMEA of Table 5.10) in the configuration and by gauging the effects of remaining time interferences by MBPTA:

G7.1. <Platform configuration>:

- The SRI crossbar supports parallel transactions between different SRI-Master and SRI-Slave peripherals. Still, there is contention when two masters simultaneously target the same SRI-Slave (i.e., on-chip memories). To ensure a fair arbitration of the simultaneous requests and prevent one partition from being starved by higher priority requests, a round-robin arbitration is established in the configuration and its maximum contention is considered in WCET analysis.
- The SPB is implemented as a shared bus that provides mutual exclusive accesses to on-chip peripherals. Each master needs to be granted bus ownership to initiate a transfer, which is arbitrated on a priority-based basis. To protect against bus starvation of lower priority masters, the AURIX includes a prevention mechanism that guarantees that all masters are granted bus access in a pre-defined period of time.
- To guarantee temporal independence among partitions $P0$ and $P3$ in core 0, a static cyclic scheduling algorithm is implemented as shown in Figure 5.10b. This scheduling is defined at design time based on WCET estimates computed for each partition by means of MBPTA.

G7.2. <Bounded temporal interference> is ensured by means of MBPTA:

G7.2.1. <Timing analysis>: The impact of relevant variability sources in the AURIX (e.g., cache) are captured by MBPTA using software randomization during execution time measurements. Time randomization in the AURIX is applied using the TASA variant (KVM+16) that has been shown to effectively handle cache-induced execution time variability in the AURIX platform (KVM+16). Bus jitter is deterministically upper-bounded offline by monitoring the number of requests and applying a contention model (DFK+17).

As an illustrative example of the application of MBPTA in the automotive case study, the pWCET of the safety-critical ‘Signal Acquisition’ ($P0$), ‘Monitoring’ and ‘Speed Control’ ($P1$) functions are illustrated in Figure 5.12 and Table 5.9

5.3 Automotive Safety Concept

provides WCET values at relevant exceedance thresholds¹.

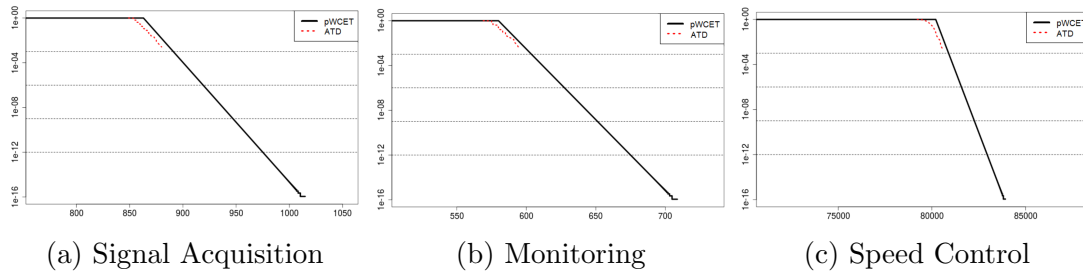


Figure 5.12: MBPTA application in the automotive case study.

The application of MBPTA to the automotive functions led to extremely tight results as, when compared to their respective HWM, the predicted pWCET bounds are always below the iconic 20% margin. The low distances for higher exceedance thresholds can be partially ascribed to the overall high predictability of the execution platform. In spite of the high determinism of the architecture, the execution time observations reflect a variability that ranged up to approximately 5% for the same program path of the three application tasks as determined by the different program layouts randomly-generated with TASA. This kind of variability, mastered by MBPTA, is typically neither controlled nor taken into account by traditional, static and measurement-based, WCET analysis procedures.

Table 5.9: pWCET at relevant exceedance thresholds for automotive case study.

Platform	HWM	10^{-3}	%	10^{-6}	%	10^{-9}	%	10^{-12}	%
Signal Acq.	880	891	1.25	919	4.43	947	7.61	975	10.80
Monitoring	595	603	1.34	627	5.38	651	9.41	674	13.27
Speed Control	80554	80888	0.41	81574	1.27	82260	2.12	82946	2.97

¹The implementation and timing analysis of this case study were conducted as part of the FP7-PROXIMA project and these results are external to this Thesis ([KCM+16](#); [AAS+16](#)).

5.3.6 Detailed Tables

This section contains the tables that support the automotive safety concept.

Table 5.10: Simplified Failure Mode and Effect Analysis for automotive case study (partitions).

Failure Mode	Failure Effect	Potential Cause	Multicore resource AURIX	Failure detection and compensation (MxCPU)
Not Executed	Safety function not performed.	Random HW fault		<Diagnostic mechanisms>.
		Design fault		<Fault prevention> to reduce the probability of systematic faults.
		Integration fault		
		Blocking execution by another partition	Core, SRI, SPB	<Temporal independence> supported by <Timing analysis> and <Platform configuration>.
Incorrect (functional) Execution	Safety function performed with wrong data, incorrect outputs may be computed.	Memory / I/O corruption (random)		<Diagnostic mechanisms>.
		Memory corruption by contending partition	Core-local scratchpad, shared SRAM	<Spatial independence> by Address Protection Sets (APS) and write protected address ranges in the <Configuration>.
		Memory corruption by components with bus master rights	Interrupt system, DMA	<Spatial independence> by write access memory protection.
		I/O corruption by contending partition	Addressable devices	<Spatial independence> by write access protection in SRI / SPB.
Incorrect (temporal) Execution	Safety function does not compute its outputs on time and may cause	Random HW fault		<Diagnostic mechanisms>.
		Blocking execution by another partition	Core, L1 caches, SRI, SPB, FPU	<Temporal independence> supported by <Timing analysis> and <Platform configuration>.

	dangerous deadline overruns.	Incorrect allocation of execution time		Impact of runtime SoJ captured in WCET by means of MBPTA and TASA.
		Blocking execution by inter-communication	SRI, shared SRAM	<Inter-partition communication> by protected shared memory regions.
		Integration fault		<Compliant item> integration tests.
'Write' access to non- assigned memory or I/O	Safety peripherals commanded by incorrect partition. Data written by safety partition may be invalidated.	Memory / I/O corruption (random)		<Diagnostic mechanisms>.
		Memory corruption	Shared SRAM, addressable devices	<Spatial independence> by write access protection in SRI / SPB.
Unable to write in assigned memory or I/O	Safety partition outputs not consistent with expected value.	Memory / I/O corruption (random)		<Diagnostic mechanisms>.
		Blocking shared resource	SRI, SPB	<Platform configuration> prevents resource starvation and locking.
Multiple combinations of previous failure modes	Uncontrolled / unpredictable behavior of safety partition.	Partition resets core and / or other core(s)		Protected <Configuration> registers.
		Uncontrolled or not repeatable initialization, configuration, startup		Protected <Configuration> registers and <Diagnostic mechanisms>.

Table 5.11: Automotive system measures for runtime error control.

Subsystem	Technique	IEC 61508-7	Maximum DC	Description
Processing units	Hardware redundancy: dual core lockstep configuration	A.1.3	High	Safety Partitions allocated to AURIX's lockstep cores. Lockstep Comparator Logic (LCL) with self monitoring for runtime HW comparison of the main and checker cores.
Invariable memory	Signature of a double word	A.4.4	High	Inclusion of a 32 bit CRC in memory.
	Program code protection	n/a	n/a	MBIST included in the AURIX processor. The Memory Test Unit (MTU) performs ROM content verification (initialization and data integrity checking).
Variable memory	Test SRAM	A.5.3	High	Periodic test "checkboard" or "march". MBIST included in the AURIX for SRAM monitoring.
Interconnect	Inspection using test patterns	A.7.4	High	'P3 DIAG' commanded cyclic test with a defined test pattern to check the correctness of on-chip buses. End-to-end data and address monitoring in SRI crossbar.
Clock	Watchdog with separate time base and window	A.9.2	High	Watchdog with upper and lower time windows and independent time base from the multicore clock.
	Over/under frequency threshold monitoring	n/a	n/a	AURIX integrates clock monitoring functionality.
Power supply	Voltage or current control	A.8.3	High	Power failure monitor that implements safety shut-off in case of power supply malfunction. Over/under-voltage control included in the AURIX.
Program sequence	Combination of timing and logical monitoring	A.9.4	High	Watchdog with upper and lower time window, only refreshed if the result of monitoring user program execution is correct. Additional core-local watchdog timers in the AURIX.
Analogue inputs	Input comparison/voting	A.6.5	High	The AURIX includes redundancy and comparison for ADCs.
Digital I/O	Test pattern	A.6.1	High	Periodic dynamic test signal monitoring.

Actuators (relay, WDG)	Monitored outputs	A.6.4	High	External actuators monitored by means of a digital input to the MxCPU that represents the state of the contact.
Partition	Independence violation detection	n/a	n/a	AURIX HW protection mechanisms and configuration enforce spatial and temporal independence (<i>G7.1/G7.2</i> of Section 5.3.5). Temporal independence is supported by WCET analysis.
Platform	BIST and start-up tests	n/a	n/a	Start-up tests and AURIX BIST during platform initialization. At operation further periodic BIST are executed.

Table 5.12: Automotive system reaction to errors.

ID	Error Group	System Reaction	Final State
SR1	Safety partition [P0 - P1] error (as defined in Table 5.4).	<ol style="list-style-type: none"> 1. P3 DIAG does not receive confirmation from faulty safety partition P0 or P1. 2. DIAG does not refresh WDG. 3. MxCPU reset. 4. MxCPU reset de-energizes safety-relay. 	Safe state
SR2	P3 DIAG partition error (as defined in Table 5.4).	<ol style="list-style-type: none"> 1. DIAG does not refresh WDG. 2. MxCPU reset. 3. MxCPU reset de-energizes safety-relay. 	Safe state
SR3	Non-safety partition (P2) error (as defined in Table 5.4).	<ol style="list-style-type: none"> 1. DIAG does not receive confirmation from faulty partition. 2. DIAG re-starts the faulty partition. 	Safety partitions remain operational
SR4	AURIX platform error detected by diagnostics.	<ol style="list-style-type: none"> 1. Error detected by diagnostics. 2. DIAG does not refresh WDG. 3. MxCPU reset. 4. MxCPU reset de-energizes safety-relay. 	Safe state
SR5	AURIX platform error undetected by platform diagnostics.	<ol style="list-style-type: none"> 1. WDG is not refreshed. 2. WDG reaches time-out and resets MxCPU 3. MxCPU reset de-energizes safety-relay. 	Safe state
SR6	Error in external WDG.	<ol style="list-style-type: none"> 1. Safety function remains operational. 2. Error detected by DIAG partition. 3. MxCPU reset. 4. MxCPU reset de-energizes safety-relay. 	Operational / Safe state
SR7	Random error in safety-relay.	<ol style="list-style-type: none"> 1. Error detected by DIAG partition (periodic test by dynamic signal generation and read back). 2. DIAG does not refresh WDG. 3. MxCPU reset. 4. MxCPU reset de-energizes safety-relay. 	Safe state

5.4 Summary

This chapter evaluates the certification argument of Chapter 4 through the elaboration of two safety concepts for the railway and automotive domains respectively. We use two different approaches, a hypervisor based solution and an approach based on hardware protection mechanisms embedded in the processor architecture, respectively. The assumptions and analysis considered at this design stage have been reviewed and positively assessed by a certification body.

5.4 Summary

Still, one key open subject is the method to determine WCET bounds of each software application to ensure temporal independence in an efficient way (without excessive resource over-provisioning). Although defined safety mechanisms factor in the impact of software timing faults in the system by switching to a safe state in case of timing overrun, assessing the correctness of the timing bounds is still crucial to preserve system's availability. The WCET estimation of multicore processors is highly complicated by the presence of complex architectural features and shared resources that result in inter-task resource contention interferences. As a consequence, several limitations arise regarding predictability, timing analysis and providing sufficient evidence to confirm that timing requirements are met in the access to hardware shared resources. In this chapter this limitation is addressed by the use of the novel MBPTA approach that provides, by construction, evidence to quantify the probability of timing overruns. However, the novelty of MBPTA entails an eminent cultural change with respect to current certification practices. This is the main subject of next chapter.

Chapter 6

Fitting Execution-Time Exceedance into Safety Standards

The safety concepts of the previous chapter rely on MBPTA technology to estimate probabilistic execution time bounds. To some extent, the concept of determining exceedance probabilities for execution time bounds may seem counter-intuitive in CRTES domains where avoiding design faults is of paramount importance. However, the reality is that probabilistic modeling is a close match to functional safety standards' philosophy to handle random hardware faults, by determining the failure rates of hardware elements and considering design techniques that make their probability sufficiently low. IEC 61508 and its child standards, like automotive ISO 26262 or railway EN 5012x, define different procedures for the management of deterministic design faults (i.e. *systematic* faults) and unpredictable hardware faults (i.e., *random* faults). They all use cognizant assessment, based on judgment from practical experience to guarantee that systematic faults are duly mitigated or controlled in such a way that their contribution to the *residual risk* of the system is acceptably low. Conversely, random faults can only be controlled at run time, and IEC 61508 requires their likelihood of occurrence to be quantified and assessed against reference values to assert with sufficiently high confidence that the necessary risk reduction is achieved. While for hardware parts the standard contemplates both systematic and random hardware faults, software faults are all deemed systematic.

The chapter proposes the quantification the likelihood of execution-time exceedance events and relating it to target failure metrics in support of certification arguments – much like for hardware random faults – instead of trusting judgment from practical experience (which is the current practice for managing systematic faults). To this end, we use MBPTA to quantify, constructively, the failure rates resulting from the likelihood of execution-time exceedance events. Earlier work describes how to design *MBPTA-friendly* hardware and software plat-

forms (KQA+16; KVM+16) such that execution-time exceedance occurs with an (arbitrarily low) probability. So far however, there is lack of understanding of how the probabilistic treatment of execution-time exceedance events can be understood by safety certification standards. Accordingly, we survey the management of systematic and random faults in IEC 61508 and propose an asymmetric treatment of software faults, addressing execution-time exceedance with MBPTA. We build on IEC 61508 metrics such as failure rates and diagnostic coverage to determine the adequacy of the design with respect to timing, much like residual hardware faults.

The rest of this chapter is organized as follows. We summarize how IEC 61508 deals with hardware and software faults in Section 6.1. In Section 6.2 we introduce the concept of execution time exceedance probability in the quantification of safety related parameters. Section 6.3 evaluates MBPTA and the hardware/-software modifications it requires with respect to the safety life-cycle defined in IEC 61508. Section 6.4 evaluates other domain-specific standards. Section 6.5 provides experimental support evidence on the railway case study. Finally, Section 6.6 presents a summary of the chapter.

6.1 Systematic and Random Faults in IEC 61508

IEC 61508 requires to provide evidence of the absence of unreasonable risk due to hazards caused by the malfunction of E/E/PE systems. To this end, IEC 61508 deals in a systematic manner with the activities necessary to develop a safety related system by adopting the safety life-cycle illustrated in Figure 2.2 of Chapter 2. Figure 6.1 provides an schematic view of the system analysis and definition phases (i.e., system specification), and realization activities of such life-cycle where the asymmetric treatment of systematic and random faults is manifested.

- **System specification:** IEC 61508 process starts by determining the scope of the system and by conducting a hazard and risk analysis to define the set of hazardous events and situations foreseeable for the system under analysis. Then, an evaluation of the risk associated to these hazards shall be performed, based on the severity, probability of exposure and controllability of the event. The fundamental goal of the standard is then to reduce this risk down to tolerable rates by the formulation and implementation of safety functions with associated SIL levels based on the level of risk. These safety functions are documented in the safety requirement specification file which is often complemented with the definition of a safety concept.
- **Realization:** The realization phase is divided into system, hardware and software development processes that define the activities to design, build and

6.1 Systematic and Random Faults in IEC 61508

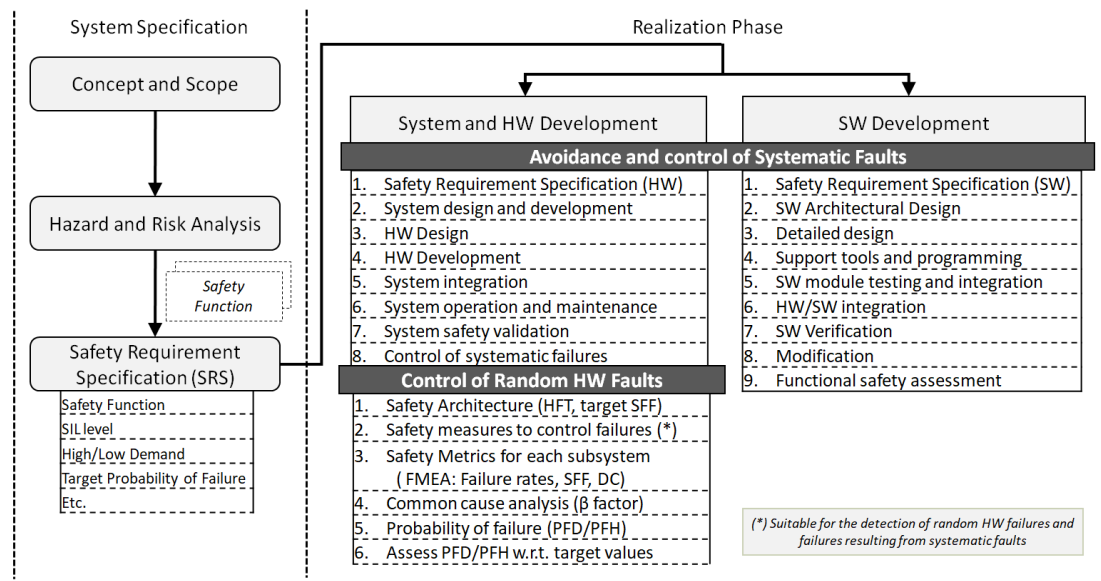


Figure 6.1: Schematic view of IEC 61508 treatment of systematic and random faults in the life-cycle.

validate the safety-related E/E/PE system.

6.1.1 Software Faults

IEC 61508 holds a deterministic view of software faults and classifies them all as systematic. Moreover, IEC 61508 assumes that all systematic faults have to be controlled, tolerated or prevented during the development process. The software development process is similar to the hardware one (Figure 6.1), except that the software development does not include a quantitative analysis of its possible faults.

IEC 61508 states that “*the probability of occurrence of systematic faults cannot in general be quantified*”. IEC 61508 sustains this assertion by arguing that, due to the nature of systematic faults, they can not be predicted with sufficient accuracy. For instance, the effects of systematic faults depend on the life-cycle phase in which they are introduced and the effectiveness of the prevention measures (e.g., structured programming) is difficult to quantify. However, IEC 61508 allows to consider that the target failure reduction for a safety function is achieved by demonstrating compliance to all requirements of the standard. In this regard, the standard introduces SIL-dependent quality management procedures for preventing systematic faults in each phase of the software development process. In practice, however, process-oriented solutions cannot provide positive evidence of the lack of residual faults, especially with the complexity of nowadays software functions,

and the hardware-software interaction that occurs in them. For this reason, the standard does also reckon runtime mechanisms suitable for the detection of errors that are the result of systematic faults.

6.1.2 Hardware Faults

As opposed to software faults that are all deemed as systematic, hardware faults are differentiated between systematic and random. Resultantly, the hardware development process demands (i) measures to prevent systematic faults, (ii) measures to control both systematic and random faults, and (iii) determination and calculation of safety related parameters to assess the suitability of the design with respect to reference threshold values.

The measures to avoid systematic faults are equivalent to those defined in the software development process, based on quality management measures during the different hardware life-cycle phases sketched in Figure 6.1. The importance to apply specific measures is dependent on the SIL level and their effectiveness is qualitatively ranged as high, medium or low. Still, IEC 61508 recognizes that there is a residual probability of systematic faults occurring resulting from the design, environmental stress or external influences or operator mistakes. Despite their probability not being quantified, the standard proposes a set of methods to control them at runtime (e.g., diversity, failure detection by on-line monitoring).

For hardware random faults instead, the procedure starts by the definition of the safety architecture and determination of associated target architectural metrics (e.g., HFT, SFF) as shown in the left side of Table 6.1 for complex components. For the calculation of safety related parameters, faults are classified as safe or dangerous faults, with the latter being further differentiated among detected or undetected faults. Safe and dangerous detected faults can be regarded as safely-ignorable since their effects become “visible” before they can do harm, or they are simply harmless. The fraction of safely-ignorable faults with respect to the overall failure rate is given by the SFF while the percentage of detected faults is given by the Diagnostic Coverage (DC). Depending on the HFT achieved by the safety architecture of the subsystem, the required SFF varies, which can only be improved either by high design effort or by increasing the portion of detected dangerous failures by the application of safety techniques and measures with appropriate DC. The SFF and DC parameters are commonly derived from an FMEA where the basis for the calculations are the failure rates of the components, that is, the average number of faults over time. In addition, for redundant components composed of two or more channels, the standard requires to determine the effect of common cause failures affecting to both channels. IEC 61508 acknowledges that diagnostic techniques cannot achieve full coverage for all types of faults and defines mechanisms with diagnostic coverages between 60% and 99%. The system may

6.2 Execution Time Exceedance Rates

therefore be exposed to dangerous undetected faults, which results in a residual risk that needs to be assessed by the quantification of the safety related metrics shown in the right side of Table 6.1: the average Probability of Failure on Demand (PFD_{avg}) and Probability of Failure per Hour (PFH).

Table 6.1: IEC 61508 quantification metrics (IEC61508).

SIL	Safe Failure Fraction (SFF)			Target probability of Failure	
	HFT 0	HFT 1	HFT 2	PFD_{avg}	PFH
4	not allowed	$\geq 99\%$	$\geq 90\%$	$\geq 10^{-5}$ to $< 10^{-4}$	$\geq 10^{-9}$ to $< 10^{-8}$
3	$\geq 99\%$	90% to $< 99\%$	60% to $< 90\%$	$\geq 10^{-4}$ to $< 10^{-3}$	$\geq 10^{-8}$ to $< 10^{-7}$
2	90% to $< 99\%$	60% to $< 90\%$	$< 60\%$	$\geq 10^{-3}$ to $< 10^{-2}$	$\geq 10^{-7}$ to $< 10^{-6}$
1	60% to $< 90\%$	$< 60\%$	n/a	$\geq 10^{-2}$ to $< 10^{-1}$	$\geq 10^{-6}$ to $< 10^{-5}$

To assess whether the residual risk (and hence the implemented measures to control faults) is acceptable, IEC 61508 defines strict pass/fail reference figures for these metrics depending on the SIL level and the mode of operation¹ of the system (Table 6.1).

6.2 Execution Time Exceedance Rates

The increasing complexity of modern computing platforms threaten the soundness of the *qualitative* assessment of timing correctness, and may allow execution-time exceedance situations to escape prevention. Execution-time exceedance may result from hardware/software interactions that are often too remote from the user reach and too difficult to control and prevent, for instance, by the combination of specific task interleaving or initial cache states. Accordingly, system's timing design can benefit from treating execution-time exceedance events much like random hardware faults, by considering the former too in the quantification of safety functions' probability of failure.

To this end, we introduce the concept of *timing failure rate*, λ_T , in the quantification of safety related parameters. The probability of failure (PFD/ PFH) of a safety function is determined by calculating and combining the probability of failure for all the subsystems which together implement the safety function (e.g., sensor, logic, actuator). The probability of failure of these subsystems is time dependent and characterized by their dangerous failure rate. This is reflected in Equations 6.1, 6.2, and 6.3 from IEC 61508-6 (IEC61508) for fail safe systems assuming a single-channel architecture with diagnostics:

¹ PFD_{avg} metric is used for safety functions that are only performed with a low demand frequency (i.e., less than once per year). For safety functions with a high operating demand or continuous mode of operation the equivalent PFH metric is used.

$$PFD(t) = \lambda_{DU} \times t \quad (6.1)$$

$$PFD_{avg}(T_{PTI}) = \frac{1}{2} \times \lambda_{DU} \times T_{PTI} \quad (6.2)$$

$$PFH = \frac{PFD(t)}{t} \quad (6.3)$$

where T_{PTI} stands for the proof test interval¹ and λ_{DU} relates to the portion of dangerous undetected (hardware) failures. Detected dangerous failure rates (λ_{DD}) are not considered in these equations as it is assumed that the diagnostic measures will detect them and switch to a safe state within the process safety time, i.e., before the fault can become dangerous (IEC 61508-2 §7.4.4.1.4).

We adapt PFD/PFH definition by considering the *probability of timing failure* as an additional subsystem in the safety loop. As opposed to hardware metrics, the probability of timing failures is constant over time and it can't be improved by proof tests and component repairs. Accordingly, we break down the PFD/PFH of the safety loop into (i) the application of Equations 6.1, 6.2, and 6.3 for dangerous undetected hardware failure rates ($\lambda_{DU_{HW}}$) and, (ii) the addition of the dangerous undetected timing failure rate (λ_{DU_T}) constant over time:

$$PFD(t) = \lambda_{DU_{HW}} \times t + \lambda_{DU_T} \quad (6.4)$$

$$PFD_{avg}(T_{PTI}) = \frac{1}{2} \times \lambda_{DU_{HW}} \times T_{PTI} + \lambda_{DU_T} \quad (6.5)$$

$$PFH = \frac{\lambda_{DU_{HW}} \times t}{t} + \lambda_{DU_T} \quad (6.6)$$

To relate the λ_{DU_T} with the overall timing failure rate (λ_T), first the portion of dangerous timing failures shall be determined. The division among dangerous (λ_{DT}) and safe failures (λ_{ST}) is usually derived from a detailed analysis of each failure mode of the subsystem (e.g., FMEA). However, IEC 61508 generally accepts a 50% division for complex components with uncertain failure modes (Part-6, Annex C). In the case of timing, it is not easy to a priori determine whether an execution time exceedance event will result in a dangerous failure of the safety function as its propagation depends in many other system level factors as the coexistence with other tasks or its impact in shared hardware resources. Accordingly, we apply this assumption to the timing failure rate to obtain the dangerous fraction of failures:

$$\lambda_{DT} = \frac{1}{2} \times \lambda_T \quad (6.7)$$

¹The proof test interval (PTI) is the maximum period of time between two proof tests that seek to diagnose undetected dangerous hardware failures, to repair the system when required and guarantee that the $PFD(t)$ does not exceedingly increase in the course of time.

6.3 MBPTA Adherence to IEC 61508

The diagnostic coverage for timing events (DC_T) then determines the fraction of detected timing failures (λ_{DD_T}) with respect to all dangerous timing failures:

$$\lambda_{DD_T} = DC_T \times \lambda_{D_T} \quad (6.8)$$

Given that $\lambda_{D_T} = \lambda_{DD_T} + \lambda_{DU_T}$, we can replace λ_{DD_T} in Equation 6.8:

$$\lambda_{DU_T} = \lambda_{D_T} - DC_T \times \lambda_{D_T} \quad (6.9)$$

From the combination of Equation 6.7 and Equation 6.9 yields:

$$\lambda_{DU_T} = \frac{1}{2} \times \lambda_T \times (1 - DC_T) \quad (6.10)$$

Subsequently, Equation 6.10 can be integrated in Equation 6.5 or in Equations 6.3 and 6.4, to contemplate execution time exceedance rates in the *PF**D* or *PF**H* metrics:

$$PF\bar{D}_{avg}(T_{PTI}) = \frac{1}{2} \times (\lambda_{DU_{HW}} \times T_{PTI} + \lambda_T \times (1 - DC_T)) \quad (6.11)$$

$$PF\bar{H} = \lambda_{DU_{HW}} + (\frac{1}{2} \times \lambda_T \times (1 - DC_T)) \quad (6.12)$$

The resulting probability is evaluated with respect to target system probability of failure metrics (defined in Table 6.1) together with other (hardware) fault modes to determine whether the system design is adequate for the integrity level assigned to the safety function.

6.3 MBPTA Adherence to IEC 61508

MBPTA provides by-construction evidence to quantify the probability of execution-time exceedance events by the determination of a pWCET distribution as that introduced in Figure 2.7 of page 25 (Chapter 2). MBPTA's viability for industrial use in safety-related systems relates to the cost of the required hardware or software changes, and how the approach can be fitted in the overall IEC 61508 safety life-cycle.

6.3.1 MBPTA Application and Feasibility

As introduced in the Background section, MBPTA requires the system to exhibit a probabilistic – hence probabilistically analyzable – timing behavior. To this end, previous work presents hardware (HAC+17; COBHAM) and software (KQA+16;

KCQ+13) solutions to implement the required time upper-bounding and time-randomization as described in Section 2.1.3. This section evaluates the safety implications of those approaches.

Hardware Modifications

Every hardware implementation that targets safety-related applications must adhere to the design guidelines determined by the certification requirements for the application domain. A solution to implement time upper-bounding and time randomization as required by MBPTA is introducing the required modifications at the hardware level. The required modifications only affect to the non-functional part of the system, in particular to the timing behavior. MBPTA hardware modifications neither change the regular hardware design flow nor the target manufacturing technology. Hence, MBPTA-compliant hardware does not involve major additional certification efforts with respect to the conventional design process.

Time-upper bounding modifications are implemented in such a way that they can be enabled during analysis and disabled during operation to avoid decreasing average performance. For instance, the hardware feature that allows enforcing the highest latency in FPU can be enabled or disabled by setting the corresponding configuration register accordingly. Similarly, the hardware is modified to cause arbitration to occur across all potential contenders regardless of whether they have pending requests or not, and, after selection of a contender, the bus is kept busy for the highest request latency. Selectively disabling this feature allows operation-time to experience fewer stalls than considered for WCET analysis. Accordingly, time upper-bounding has no influence in the executable at runtime.

On the contrary, time randomization must be kept enabled during both, analysis and operation phases, since this is the only way to guarantee that observations during analysis match (probabilistically) those during operation. Bus protocols like AMBA (AMBA) (one of the most, if not the most, used), do not define any particular arbitration policy. This situation allows adding random arbitration policies with no impact on the protocol specification. The same happens for cache placement and replacement. While the latter is already supported in many processors, adding the former requires combining the address being accessed with a random number, changed across runs, to map the address to a random cache set. This change causes the timing behavior of cache conflict scenarios that are probabilistically relevant to be close to average behavior which, in turn, is very close to the typical behavior on conventional hardware designs. However, all components building upon time randomization, if implemented in hardware, require a *Source of Randomization (SoR)* also in hardware. This has been achieved by implementing a suitable *PRNG* that provides a sequence of random numbers presented in Chapter 7. The very fact that a failure in this SoR impacts the timing behav-

ior of safety functions at runtime, causes it to become a safety-related item, with corresponding safety-related requirements.

Software Modifications

COTS processor's hardware cannot be customized as proposed in the previous approach. As an alternative, time upper-bounding can be applied offline by monitoring relevant events and padding execution time observations accordingly. For instance, for the FPU, the highest latency of floating point operations is multiplied by the number of executed floating point operations, obtained by means of PMCs. Similarly, bus jitter can be deterministically upper-bounded by monitoring the number of bus access requests with PMCs and applying a contention model that assumes worst-case overlap among them (DFK+17).

For randomizing the timing behavior of COTS architectures, software solutions have been developed to apply modifications either dynamically (KCQ+13) or statically (KQAF+14; KVM+16). The former, DSR, introduces randomization at runtime, by placing memory objects dynamically in random locations in memory every time the program is invoked and/or when objects are created. This requires the compiler to introduce some runtime randomization code (indirections through pointers) in the program executable. The latter, SSR, achieves the required randomization effect in an entirely static manner by generating several binaries, with different (randomly generated) memory layouts, for the same program. SSR can be implemented at linker level (KQAF+14) or alternatively, at source code level (KVM+16). In the former, the linker randomly places functions in the binary while the latter directly randomizes the location of memory objects by using their definition in the source-code of the program.

DSR and linker level SSR require modifications at compiler-linker level. While they only alter the regular software development process by introducing an additional compiler pass, considering a compiler that lacks of in-service evidence in the safety-critical industry and that modifies the binary at runtime, challenges certification. On the contrary, the static source code level randomization, referred to as Toolchain-Agnostic Software Randomization (TASA), requires no changes in the system standard stack toolchain and is hence, the most promising solution from a certification viewpoint, as it can be used with regular (qualified) compiler-linker tools. However, the generation of multiple binaries is not contemplated in the safety software lifecycle. For the application of MBPTA with TASA the binaries shall be used as follows (Figure 6.2):

- N automated executions with statistically significant number of N different TASA-generated binaries are used in the timing analysis phase.

- The execution time observations of the N different TASA-generated binaries are fed to MBPTA to derive a pWCET estimate. The resulting pWCET curve is probabilistically representative of the timing events of each particular (TASA-generated) binary.
- Then, only one of those binaries, with a given memory layout, is deployed in the final system. This binary, at operation time, meets the MBPTA requirement of representativeness, as the corresponding extreme timing behavior fits the envelope computed by MBPTA, since the probability of exceedance determined by the use of TASA is equivalent to the execution-time exceedance probability of all systems with the same randomly-generated binary.
- Functional tests are performed only on the deployment-time binary, as in the regular development process, except for the verification of timing properties where MBPTA is used with N binaries as depicted in Figure 6.2. There is no need to functionally test the other binaries as (i) they are not destined to operation, and (ii) they are functionally identical to the one being verified.

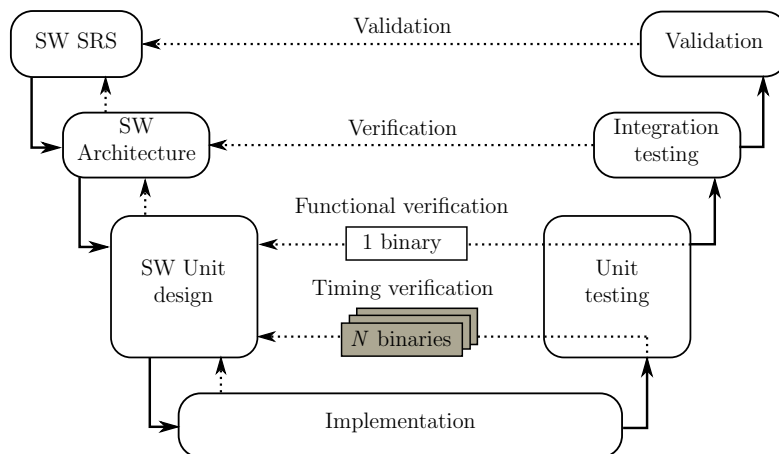


Figure 6.2: Implications of SSR on a typical development cycle.

6.3.2 MBPTA in IEC 61508 Safety Life-cycle

Coming back to IEC 61508 development process and with emphasis in timing-related requirements, next we describe how the MBPTA approach can fit within the software development process defined in the standard (Figure 6.3). MBPTA activities relate to IEC 61508 software development cycle as follows:

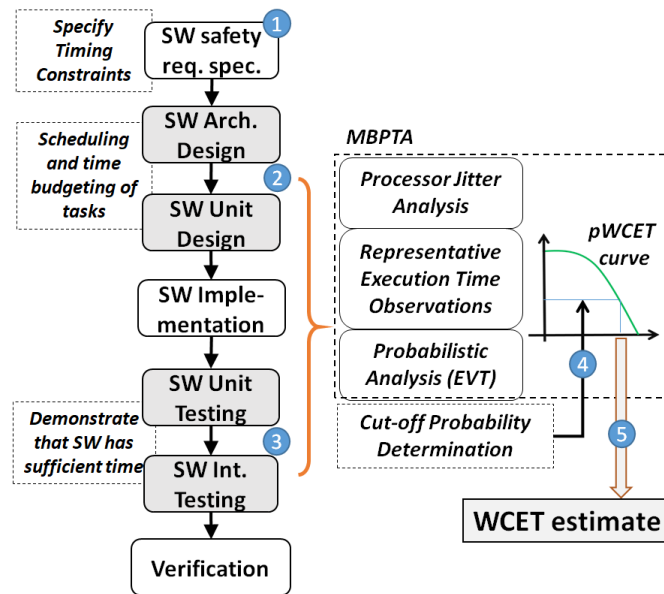


Figure 6.3: Sketch of how MBPTA fits in IEC 61508 software development process.

1. IEC 61508 requires that system designers specify the time budgeting of critical software in the software requirement specification phase. MBPTA results shall be compared with respect to these bounds in the verification phase to assess the suitability of the designed system with respect to the requirements.
2. Software design requires adopting safety measures for the avoidance of systematic faults, including cyclic scheduling regimes that rely on worst-case execution time analysis techniques for the dimensioning of the time slots.
3. During unit and integration testing, the standard requires evidence that the software is allocated enough time to complete its functionality. This implicitly requires controlling all low-level hardware/software sources of execution-time jitter during the analysis process to assure that the obtained WCET estimation does upper-bound, as tightly as possible, the real WCET.
4. Covering this requirement by means of MBPTA, results in a pWCET distribution rather than in a single WCET value. Accordingly, the designer needs to derive the appropriate cut-off exceedance probability (or execution time exceedance rate per hour). In this chapter we propose deriving this bound from the system tolerable failure rate, evaluating it together with the diagnostic coverage for timing errors and the SIL of the safety function.
5. Finally, with the cut-off probability, the corresponding WCET is extracted from the pWCET curve.

6.3.3 Cut-off Probability Determination

The cut-off probability of execution time exceedance can be determined from Equations 6.11 and 6.12 of Section 6.2 based on the reference *PFD* or *PFH* values defined in Table 6.1. The hardware and dynamic software solutions, attain randomization at program run granularity. Accordingly, the pWCET distribution provides the execution time exceedance probability *per execution* of the task. In order to relate the cut-off probability of exceedance with the timing failure rate (λ_T) per hour or Failures In Time (FIT), the pWCET exceedance probability (EP_{pWCET}) shall be multiplied by the task's execution frequency per hour (F_{UoA}):

$$\lambda_T = EP_{pWCET} \times F_{UoA} \quad (6.13)$$

For instance, to obtain a timing failure rate of 1 FIT (10^{-9}), if the safety function is executed 10^3 times per hour, the system designer should choose the WCET estimate computed for the 10^{-12} exceedance threshold. In this way, it is probabilistically guaranteed that the accumulated execution time exceedance rate of all instances of the program executed per hour is below 10^{-9} .

As opposed to the hardware and dynamic software solutions, static software randomization (i.e., TASA) introduces randomization per binary. As a result, the probability of exceedance determined by the use of TASA is equivalent to the execution-time exceedance probability of all systems with the same randomly-generated binary. For redundant systems, using a different TASA-generated binary in each of the redundant nodes would therefore reduce the probability of experiencing execution time exceedance events in all nodes. However, this approach implies each unit having a different binary, which may not be acceptable if each individual unit is not fully tested. If in contrast all redundant nodes are deployed with the same binary, if the binary exceeds the pWCET, it may do so in all units.

Diagnostic Coverage

As exposed in Equation 6.8, diagnostic coverage for WCET estimate overruns needs to be considered in the quantification of the probability of failure since existing safety mechanisms can factor in the execution-time exceedance's impact and prevent its escalation into a *system failure* (i.e., they can be categorized as dangerous detected failures, λ_{DDT}). The standard suggests the usage of watchdog timers to detect the consequences that a fault in a hardware component may have in the program sequence (e.g. the program not executing, too slow execution or too fast program execution). In this scenario, standards categorize the diagnostic coverage achievable by watchdogs (for errors in the control logic of processing units) as low (60%) or medium (90%). Accordingly, watchdogs can also detect (possibly with a high (>99%) diagnostic coverage) timing overruns in the operational system.

6.4 Domain-specific Standards

On the occurrence of such an event, the safety mechanisms detect the error and a proper action is taken, efficiently reducing the residual risk.

The nature of the safety function, either fail safe or fail operational, conditions the action to take after the detection of the exceedance event and has a different impact in the residual risk:

- In the case of fail safe systems, the system is moved to a safe state each time a diagnostic mechanism detects an execution time exceedance event, and hence, system's safety is preserved at the expense of making some functionality (or the entire system) unavailable. As a result, the degree of diagnostic coverage that the safety mechanisms provide for timing failures (DC_T) shall be considered when determining the cut-off probability as described in Equations 6.11 and 6.12. Whereas in such circumstances an execution-time exceedance may not compromise system safety, the MBPTA approach can improve the design by assessing the availability of the system from a timing perspective.
- In the case of fail operational systems, i.e. systems that cannot reach a safe state because they need to be operational to maintain safety, safe operation in the advent of a deadline violation can be preserved by, for instance, using redundant architectures so that the failure of one unit does not stop (safe) operation of the entire system. Whenever this is not possible, having a high diagnostic coverage against timing exceedance events is not sufficient to preserve safety and a sufficiently low cut-off probability needs to be chosen to ensure that the contribution of timing faults to the residual risk is sufficiently low.

6.4 Domain-specific Standards

Standards derived from IEC 61508, like automotive ISO 26262 or railway EN 5012x, follow the same philosophy for the management of systematic and random faults although they may include additional refinements fit for the particular domain. For instance, the fault models and calculation of the corresponding metrics used in the automotive ISO 26262 standard are more detailed than in the IEC 61508 meta-standard. ISO 26262 addresses non safely-ignorable faults by defining the *single-point fault metric* (SPFM) that determines the item's robustness to single-point and residual faults by either design or safety mechanisms, and the *latent fault metric* (LFM) that determines the item's robustness to latent faults. These metrics are evaluated against pass/fail reference intervals defined in the standard for each target integrity level. The railway domain instead defines the Tolerable Hazard Rate (THR) per hour and function, analogously to

PFH in IEC 61508. Regarding software faults, however, the approach is identical in the three reference standards: software faults are considered systematic and qualitative measures are recommended for fault avoidance, such as the support of WCET analysis to sustain temporal independence among software elements. As IEC 61508, ISO 26262 and EN 5012x determine the requirements for avoiding or controlling systematic faults based on expert judgment from practical experience and assume that they can be neglected compared with random hardware failures if necessary software engineering measures are taken.

Overall, all IEC 61508 based standards retain the notion of random hardware faults and propose a qualitative approach for software faults that may not scale well against increasingly complex systems. Arguably, therefore, all the application domains covered by the IEC 61508 umbrella might equally benefit from incorporating an execution-time exceedance quantification approach following the solution presented in this chapter.

6.5 Experimental Support Evidence

To sustain MBPTA application in safety-critical systems and the quantification of execution time exceedance rates, this section discusses an exemplary application of MBPTA in the railway case study defined in Chapter 3. We do *not* aim to present a full WCET analysis method for the target platforms, the intent of this section is just to provide an empirical evaluation of the application of MBPTA in an industrial safety-related case study and show that the execution-time jitter of hard-to-predict resources like the cache can be handled with MBPTA.

6.5.1 Collecting Analysis-time Observations

The application we consider is a subset of the railway case study introduced in Section 3.2.1, referred to as UoA. As a matter of illustration, from the 10 execution paths, we only show the results for the path with longest average execution times (path 7) and the one with shortest average execution times (path 9). For the sake of comparison, we collect execution time observations for the UoA when executed in the plain LEON3-MC and P4080 platforms and their MBPTA-compliant versions: LEON3-MC-HR, LEON3-MC-DSR and P4080-DSR.

- **Hardware Modifications:** Figure 6.4 reports the execution-time variability observed for the railway case study in the LEON3-MC platform. The left side exposes the execution time samples observed across one thousand runs of the same execution path in the MBPTA-compliant platform (LEON3-MC-HR). In the left side, Figure 6.4 presents the observations resulting from the

6.5 Experimental Support Evidence

same experiments in the baseline LEON3-MC platform, where the sources of jitter are not controlled.

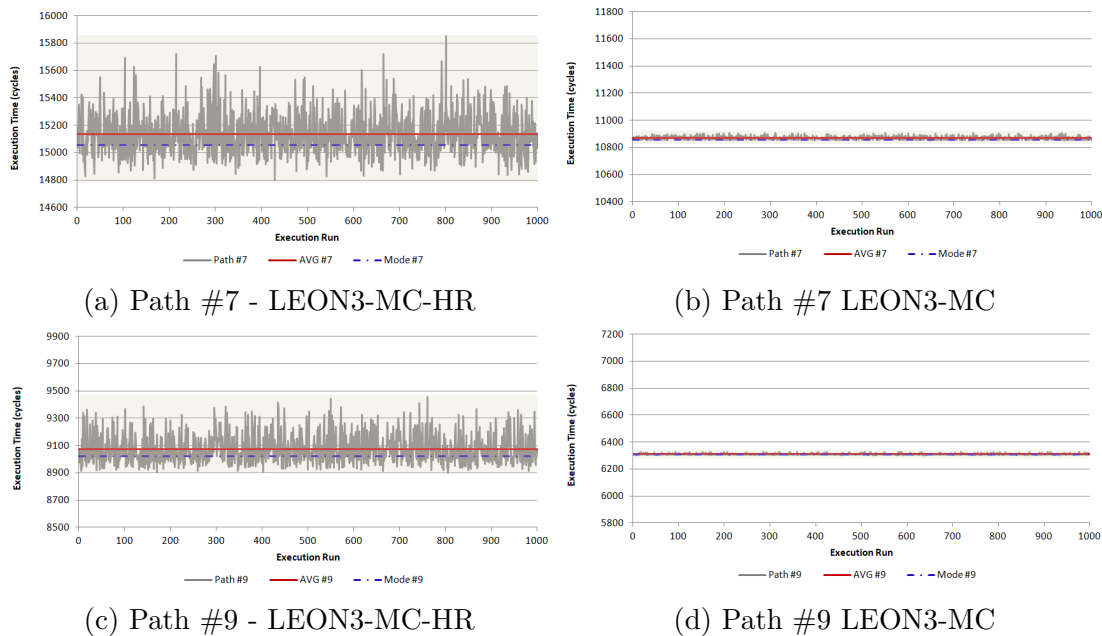


Figure 6.4: Observed execution time variability in the railway application.

Figures 6.4a - 6.4c report that the execution time variation in the time randomized architecture range up to 7% over the average while the non time randomized variation is below 1%. The increased variability in the MBPTA-compliant platform (in the range of 600-1000 cycles for all paths) represents the conditions that, at operation, may result in varying execution times. In particular, the effects of L1 and L2 cache conflicts, FPU operations latency and bus arbiter are exhibited in the analysis-time measurements. This variability is typically hardly disclosed in the measurements obtained without any control on hard-to-reach low-level hardware/software interactions. This is reflected in the results of the baseline platform where the observed variability is small (less than 60 processor cycles in all analyzed paths).

- **Software Randomization:** In Figure 6.5 we compare software randomized results in the LEON3-MC-DSR and P4080-DSR with respect to their counterparts without randomization. In particular, the impact of memory placement and cache layout is made probabilistically analyzable by means of dynamic software randomization.

We observe that execution time variation, which may also incorporate the effects of other sources of execution-time jitter, is around 3 times bigger

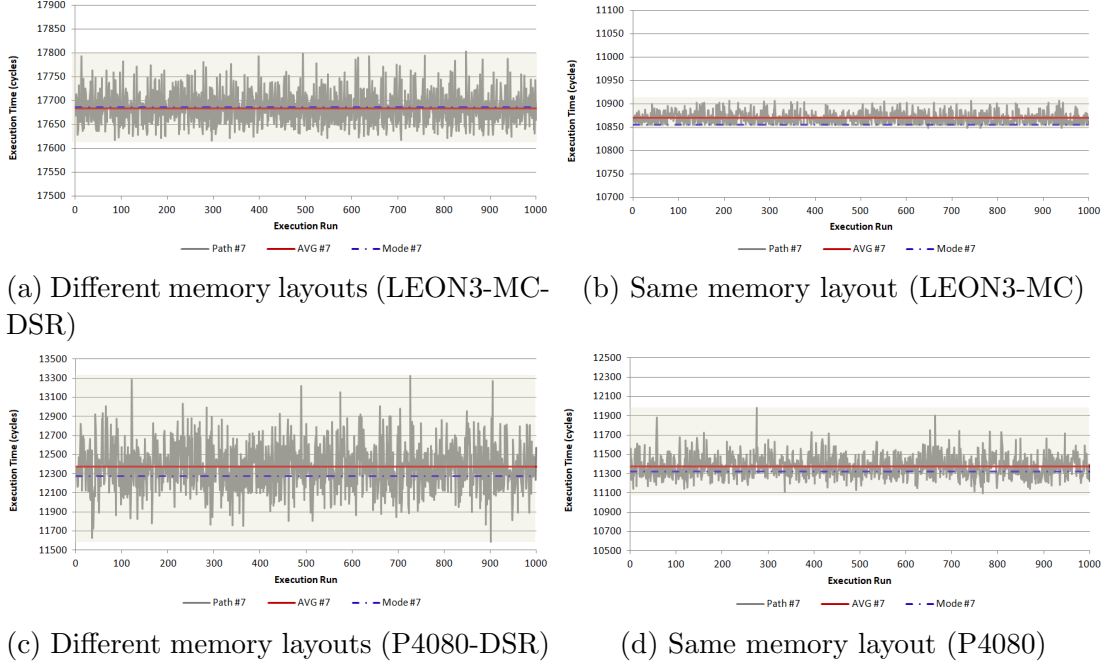


Figure 6.5: Execution time variability caused by program layout (Path #7).

in the software randomized versions of both platforms when compared to the results in the baseline architectures. This kind of variability resulting from program layout is typically hardly controlled or taken into account by traditional, static and measurement-based, WCET analysis procedures. Even more critical is the fact that classic measurement-based techniques *do not allow building arguments* on whether and to what extent the effect of jittery resources has been captured at analysis time.

Results in both, hardware and software, solutions show how MBPTA-compliant platforms deal with execution-time jitter to ensure that MBPTA outcomes are representative and hence, sound.

6.5.2 Application of MBPTA

We then applied MBPTA to the execution times observed in the three platform setups. Figure 6.6 shows the resulting pWCET for paths 7 and 9 and Table 6.2 compares the pWCET bounds at relevant exceedance thresholds against the HWM for the ten analyzed paths.

The observations collected in the MBPTA-friendly platforms successfully passed the statistical i.i.d. tests, which allowed using them as input to the subsequent probabilistic analysis process. To the latter end, we applied the EVT to the

6.5 Experimental Support Evidence

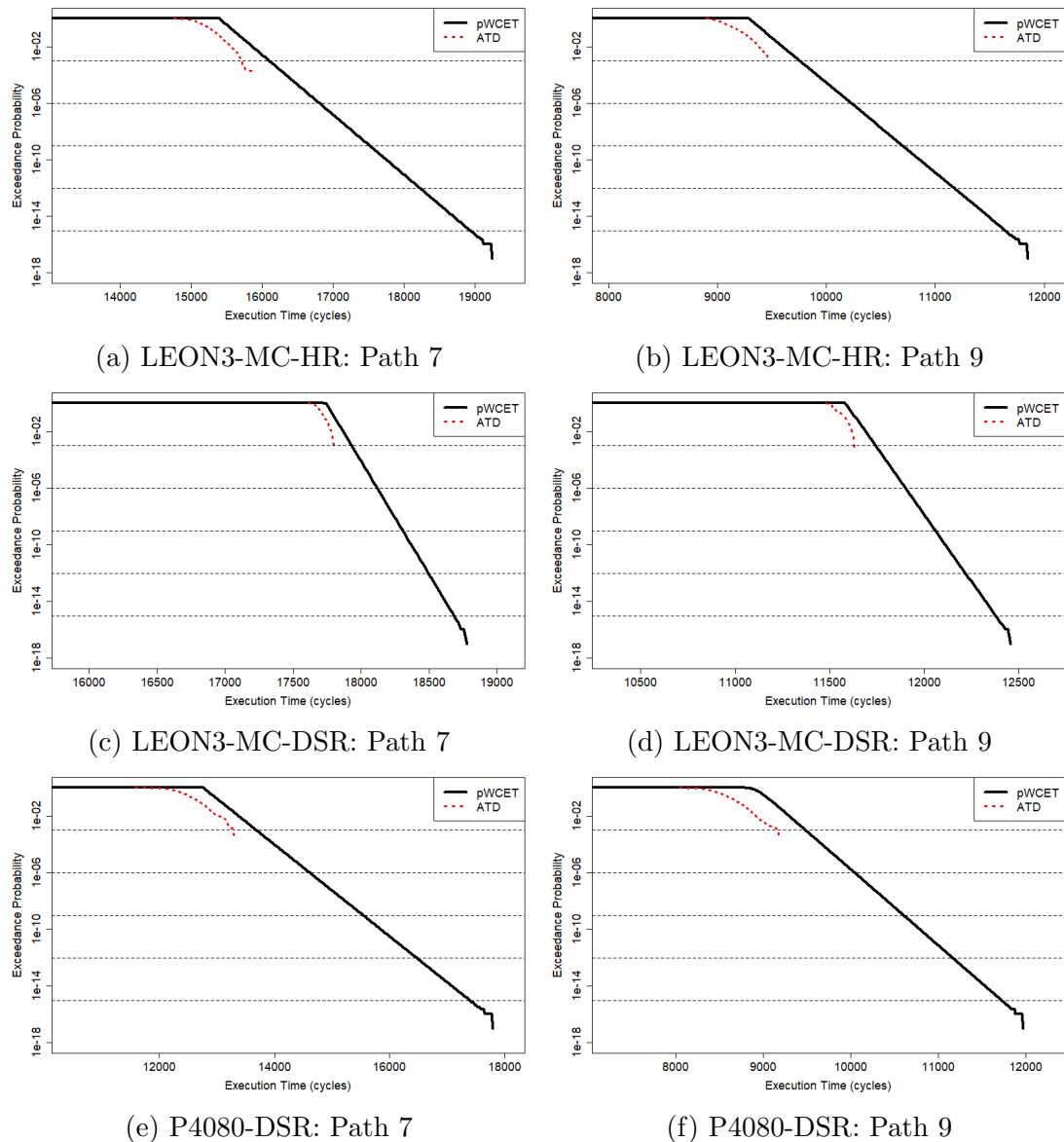


Figure 6.6: The pWCET distributions computed by MBPTA for the railway application subset path 7 (longest) and 9 (shortest).

observed execution times. In all cases, 1,000 measurements were sufficient for MBPTA to converge: adding additional observations for each application would *not* change the resulting pWCET distributions (solid black lines) shown in Figure 6.6. The red dotted line plots the analysis-time execution time observations, i.e., the ATD in the form of Complementary Cumulative Distribution Function (CCDF).

Table 6.2: pWCET bounds at relevant exceedance thresholds (in processor cycles).

	Path	HWM	10^{-3}	%	10^{-6}	%	10^{-9}	%	10^{-12}	%
LEON3-MC-HR	0	14268	14640	2.60	15366	7.70	16110	12.91	16836	18.00
	1	14386	14632	1.71	15202	5.67	15790	9.76	16360	13.72
	2	13725	13848	0.90	14396	4.89	14962	9.01	15510	13.01
	3	13397	13719	2.40	14366	7.23	15030	12.19	15678	17.03
	4	13813	14012	1.44	14506	5.02	15019	8.73	15531	12.44
	5	13616	13941	2.39	14548	6.84	15173	11.44	15780	15.89
	6	14412	14720	2.14	15415	6.96	16092	11.66	16787	16.48
	7	15850	16144	1.85	16849	6.30	17553	10.74	18258	15.19
	8	10041	10239	1.97	10803	7.59	11355	13.09	11907	18.58
	9	9543	9782	2.50	10254	7.45	10726	12.40	11198	17.34
LEON3-MC-DSR	0	16701	16744	0.26	16804	0.62	16885	1.10	16946	1.47
	1	16081	16134	0.33	16232	0.94	16310	1.42	16407	2.03
	2	15496	15592	0.62	15743	1.59	15874	2.44	16025	3.41
	3	15290	15354	0.42	15466	1.15	15577	1.88	15670	2.49
	4	15683	15815	0.84	15986	1.93	16138	2.90	16290	3.87
	5	15677	15726	0.31	15802	0.80	15859	1.16	15916	1.52
	6	16314	16381	0.41	16480	1.02	16579	1.62	16678	2.23
	7	17803	17960	0.88	18154	1.97	18348	3.06	18520	4.03
	8	12221	12294	0.60	12368	1.20	12457	1.93	12546	2.66
	9	11633	11765	1.13	11920	2.47	12075	3.80	12245	5.26
P4080-DSR	0	15729	16242	3.26	17359	10.36	18456	17.34	19552	24.31
	1	12527	12911	3.07	13850	10.56	14790	18.06	15730	25.57
	2	12548	13004	3.63	14068	12.11	15132	20.59	16196	29.07
	3	12356	12781	3.44	13673	10.66	14565	17.88	15457	25.10
	4	12328	12886	4.53	14029	13.80	15173	23.08	16316	32.35
	5	12382	12650	2.16	13471	8.80	14275	15.29	15079	21.78
	6	12237	12815	4.72	13949	13.99	15067	23.13	16202	32.40
	7	13322	13716	2.96	14639	9.89	15579	16.94	16501	23.86
	8	10290	10603	3.04	11326	10.07	12048	17.08	12771	24.11
	9	9203	9499	3.22	10061	9.32	10623	15.43	11184	21.53

Our experiments show that, limited to the processor architectures considered in this chapter, a safety margin at 20% would be conservatively pessimistic and therefore sound in some cases (e.g., LEON3-MC-DSR). Yet, for other processor architectures like P4080-DSR, that margin would be optimistic for low exceedance thresholds (10^{-9} , 10^{-12}) and hence, unsound. The slope of the pWCET distribution, and hence the margin above the highest observed value, depends on the particular characteristics of the program under analysis and how it uses the underlying processor hardware. That would be the exceedance probability if *all* upper-bounded sources of jitter always caused their highest latency. In general, this is not the case but how often this would happen cannot be told beforehand since this depends on application's input data during operation. MBPTA provides

6.5 Experimental Support Evidence

a method to strictly upper-bound the residual risk of execution-time exceedance.

Cut-off probability determination

The particular cut-off probability can be determined based on system safety metrics as proposed in Section 6.2. Considering a single channel of the emergency module of the railway case study we make the following assumptions:

- *PFH* share: The safety function is executed in high demand mode by a safety loop that includes a set of sensors, programmable electronic logic (either LEON3-MC-HR, LEON3-MC-DSR or P4080-DSR) and an actuator (emergency break). The PE logic (including timing failures) takes a 25% share of the overall failure rate (sensors and the actuator are assumed to take 30% and 45% respectively).
- $\lambda_{DU_{HW}}$: A failure analysis determines that the dangerous undetected hardware failure rate of the logic is of 18 FIT (for simplicity, same value is assumed for all three platforms).
- T_{PTI} : The proof time interval is of one year (8760h).
- DC_T : The system includes a watchdog to detect timing overruns with a high (99%) diagnostic coverage.
- F_{UoA} : The task under analysis is executed 10^3 times per hour.

We first obtain the target probability of failure for the complete safety loop from IEC 61508 (Table 6.1) and we compute the 25% that applies to the logic and timing failures:

$$PFH = 0.25 \times 10^{-7}$$

With all these data, we derive the target timing failure rate, λ_T , from Equation 6.12:

$$\lambda_T = \frac{2 \times (PFH(0.25 \times 10^{-7}) - \lambda_{DU_{HW}}(18 \times 10^{-9}))}{1 - DC_T(0.99)} = 1.4 \times 10^{-6}$$

According to Equation 6.13, as the task is executed 10^3 times per hour, the failure rate of 1.4×10^{-6} can be achieved by the WCET estimate that corresponds to 10^{-9} exceedance probability per execution (EP_{pWCET}).

6.6 Summary

Postulating that execution-time violations can all be prevented by the procedures defined in standards for systematic faults is becoming increasingly prohibitive with the use of increasingly complex hardware and software, yielding unsustainable ratios of effort versus quality of outcome. The latter question leverages the need to step up the guidelines of current safety standards to new-generation processors.

In this chapter, we have presented a quantitative approach for determining the likelihood of execution-time exceedance events – similarly to what is done for random hardware faults – to assess the corresponding residual risk of exceeding a deadline. The proposed approach relies on MBPTA, which allows deriving the WCET estimation from a probabilistic WCET distribution. We determine the cut-off probability based on the risk reduction required for the integrity level associated with the safety function. This chapter presents the approach in the context of the IEC 61508 software development process with the intent of facilitating the acceptance of execution-time exceedance rate quantification in the standard, following the procedure established for random hardware faults. In addition, we analyze the safety implications of both hardware and software modifications to achieve MBPTA-compliance in the platform. In contrast with current practice where a safety margin “assesses” (qualitatively) that the estimated WCET *cannot* be exceeded, our solution provides scientific reasoning with quantitative evidence that reduces the uncertainty.

Chapter 7

Safety-Related Pseudo-Random Number Generator

As discussed in previous chapters, MBPTA rests on the premise that the timing behavior of certain processor resources is time randomized. In particular, randomization helps achieving representativeness for those resources whose jitter is big enough to prevent them to work on their worst latency. For hardware-injected randomization, the approach on which we build, this requires a built-in source of randomization, SoR.

Some platform resources require using random data from the SoR at every few processor cycles, for example for random replacement in caches or on-chip bus arbitration. This requirement raises pressing needs for high-frequency hardware SoR and for addressing the challenges entailed in achieving trustworthy randomization. In this chapter, the requirements on hardware SoR are identified and addressed, whose satisfaction allows MBPTA to be used with safety-related software programs. As this SoR is a key component to attain the randomization required by MBPTA technology, in order to prevent introducing faults by the use of MBPTA in safety related systems, the design of the SoR shall guarantee its correct behavior. To this end, in this chapter we design an SoR that meets with the following non-functional requirements:

- **Area and Power.** Embedded systems are subject to stringent area and power constraints. A hardware SoR should therefore be implemented with low complexity, and equally low area and power budgets.
- **Randomness.** Preventing correlation among randomized events is a prerequisite to use with MBPTA ([CSH+12](#)). A hardware SoR must therefore deliver long sequences of high-quality random numbers that do not cause correlation patterns in the results, thus achieving and preserving independence.

- **Safety.** To achieve IEC 61508 compliance in mixed-criticality implementations that use a hardware SoR, its design must conform with the safety-related principles that apply to the highest SIL in the system.

To this end, in this chapter we evaluate the benefits that a hardware SoR provides as PRNG and we make four key contributions to foster the applicability of MBPTA for IEC 61508 compliant mixed-criticality systems that employ modern multicore processors:

1. We start by devising two high quality PRNG designs, the Multiply-With-Carry (MWC) and Linear Feedback Shift Register (LFSR), and propose a low-area low-power MBPTA specific PRNG implementation taking one of them as a baseline in Section 7.1.
2. Section 7.2 then analyzes the number of SoRs required for a reference multicore architecture in which the timing behavior of the caches, TLBs, on-chip bus and memory controller is randomized.
3. We study how a hardware PRNG can be made compliant with IEC 61508 SIL 3 design principles in Section 7.3.
4. Section 7.4 provides an evaluation of the proposed PRNG, showing that it achieves good randomization properties.

Finally, Section 7.5 presents a summary of the chapter.

7.1 High-Quality Low-Cost PRNG

Implementing a trustworthy random source is complex. The stringent requirements on low-power and area pose hard constraints on hardware solutions. Moreover, the need for reproducibility put forward by verification and validation processes discourages the use of true random sources. IEC 61508 remarks the importance of *repeatability* in all testing phases and so does to evaluate the adequacy of support tools (IEC 61508-3 §7.4.4). This makes PRNGs an attractive solution as an SoR. PRNGs have been studied in several fields, most notably in cryptography, to approximate the properties of sequences of random numbers (RSN+10). Whereas the generated numbers are not truly random – as they can be completely determined from the initial set of values (a.k.a. *seeds*) used to initialize the PRNG – they still show good randomness properties.

7.1 High-Quality Low-Cost PRNG

7.1.1 Sought Properties

The main properties that a PRNG must exhibit to be fit for use with MBPTA are *period* and *randomness*. In addition, even if it is not a mandatory property for a hardware SoR, PRNGs provide *reproducibility* by construction. As stated above, reproducibility is a very valuable feature for verification and validation purposes.

Period

A PRNG provides a sequence of numbers whose period must be long enough to ensure that either repetition does not occur during system lifetime or, if it occurs, it does after a long enough time for any potential correlation between the outcomes of the system at different time instants to be probabilistically irrelevant.

Randomness

Randomness should not be understood as an all-or-nothing metric since it has variable degrees. The degree of randomness for a number of bit sequences generated with a PRNG is proven statistically by checking the lack of meaningful patterns, repetitions, imbalance between different values, etc. In this way, a suitable PRNG supports the claim that the sequences of random bits produced exhibit the same properties as those sequences generated by a truly random number generator. The quality of the randomness attained by the generator can be measured with standard tests such as the one used by the US National Institute of Standards and Technology ([RSN+10](#)). Those tests are specifically designed to assess the sensitivity to certain weaknesses of the PRNG as those described before.

Reproducibility

A PRNG produces a sequence of pseudo-random numbers starting from one or several seeds. This sequence has a finite and fixed size (i.e. period) before it repeats. The particular seeds used for its initialization determine the particular sequence produced. Every time a new pseudo-random number is produced, the seed is updated automatically so as to determine the next pseudo-random number. This process repeats until the full sequence is produced and the new seed is the one set initially. Hence, PRNGs provide reproducibility as a function of the initial seed. Yet, randomness is maintained by using a random seed (e.g., generated by software means). Using different seeds leads to different starting points, and thus to uncorrelated random sequences. In this way, the starting point in the full sequence of the PRNG becomes controllably random.

The initialization of the seeds has to be done conveniently. In general, one can use software-generated random values to initialize a PRNG. A practical way

to perform that task is to delegate it to system software. System software may set the seeds on its own at boot time by writing to the addresses where seeds are mapped or prompt the user to provide values for the seed, which is useful for the sake of reproducibility. However, once the system is deployed, only system software must have the privilege to set seeds.

7.1.2 MBPTA Convenient PRNG

Unlike reproducibility, which is obtained by construction, period and randomness need to be evaluated for any PRNG as they determine the quality of the random numbers generated with it. To this end, a reference baseline is selected against which the acceptability of the PRNG can be justified. Based on the randomness tests and set of PRNGs provided by the US National Institute of Standards and Technology ([RSN+10](#)), the MWC ([MZ91](#)) and LFSR ([Alf96](#)) solutions explained below show highest pass rate and hence, best properties.

Multiply-With-Carry (MWC) PRNG

The MWC ([MZ91](#)) produces random numbers based on the following set of equations:

$$seed_z = 36969 \cdot (seed_z \& 65535) + (seed_z \gg 16) \quad (7.1)$$

$$seed_w = 18000 \cdot (seed_w \& 65535) + (seed_w \gg 16) \quad (7.2)$$

$$RII = (seed_z \ll 16) + (seed_w \& 65535) \quad (7.3)$$

where $seed_z$ and $seed_w$ are the seeds of the PRNG, $\&$ stands for a logical *AND* function, \gg and \ll stand for logical bit shifts, and RII is the random number generated. Both seeds are updated to produce a different number every time (Equation (7.1) and (7.2)).

An efficient implementation of MWC is proposed in the logical design of Figure 7.1. The proposed design comes from the observation that logical *AND*, bit shifts and 16-bit additions required by MWC are simple operations in hardware. Multiplications, which are much more complex, can be transformed into a set of few additions given that one of the operands is known and the number of ‘1’s is low. For instance, 36969 (9069h) has only 6 bits set to one, so we can transform such a multiplication into an addition of 6 16-bit numbers. Similarly, 18000 (4650h) can be transformed into an addition of 5 16-bit numbers. Thus, each seed generation requires 6 or 7 additions in total, which can be arranged in a binary tree of 3 levels of 2-input adders (Figure 7.1). The resulting RII just selects a subset of the bits of the two seeds, so it does not introduce any delay.

7.1 High-Quality Low-Cost PRNG

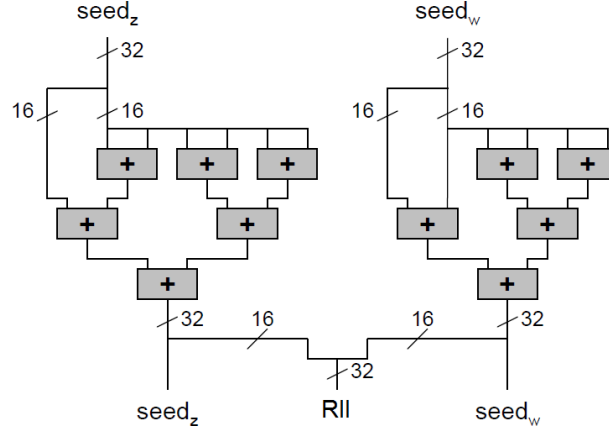


Figure 7.1: Proposed implementation of the MWC PRNG.

Linear-Feedback-Shift-Register (LFSR) PRNG

The baseline LFSR implementation produces one bit per iteration (see Figure 7.2 (a)). The LFSR consists of R bits that constitute the seed. The actual value of R determines the PRNG period and the feedback bits of the shift register. In the remainder of this chapter the value of R equals 168 as this is the largest value described in (Alf96), and accordingly, the one with the longest period before repetition. Note that the produced random bit is the result of carrying out XNOR of some of those R bits. Every time a random bit b is produced, the most significant bit of those R bits – R_{168} in the example – is discarded. The remaining bits ($R_{167} - R_1$) are shifted left, thus occupying positions $R_{168} - R_2$, and b is stored in position R_1 .

We propose changing the LFSR by replicating the XNOR process n times, which allows generating n pseudo-random bits ($b[n]$) at every cycle. The number (n) of bits that can be provided per cycle is limited by the lowest position of the bits XNORed in the 1-bit version. For instance, as such bit is in position R_{151} in the example, then up to 151 bits can be produced per cycle given that the last bit produced out of those 151 bits would use the bit in position R_1 and the next random bit will use as input one of the bits produced in this cycle. Figure 7.2(b) illustrates the parallel version of the LFSR when 32 bits are produced. As shown, XNOR gates operate on the same relative bits and produce 32 random bits ($b[32]$); bits in positions $R_{136} - R_1$ are shifted 32 positions left, and bits $b[32]$ are used to update positions $R_{32} - R_1$.

The hardware implementation of a LFSR producing 32 bits includes a register of R bits (168 in the example). The longer the register, the longer the period. Also 32 4-bit XNOR gates operating in parallel are needed. The total delay is just the addition of the XNOR gate delay and the register update.

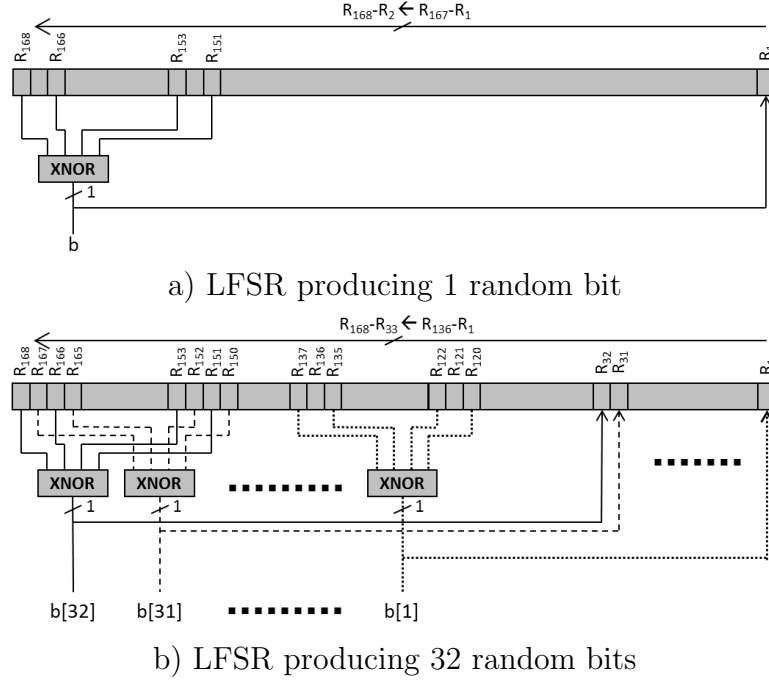


Figure 7.2: 168-bit LFSR implementation (producing 1 & 32 bits).

Comparison

As stated above, both MWC and LFSR PRNGs provide similar randomness properties, passing 187 out of the 188 tests (99.5%) proposed by the authors in (RSN+10). Despite randomness, low area, time and energy consumption features are decisive for a PRNG to be embedded in systems subject to stringent area and power constraints, even more so for multicore mixed-criticality implementations whose major benefits are in those non-functional dimensions. In terms of delay and energy overhead, both PRNGs fit in one processor cycle at 4 GHz , and incur less energy than an access to cache. When it comes to period, the LFSR solution provides particularly favorable response, with 2^{163} cycles vs. 2^{60} for the MWC. In other words, assuming a processor operating at 1 GHz and the PRNG delivering one random number per cycle, the random number sequence generated with a LFSR would take 3.7×10^{32} years to repeat while the one generated with the MWC would repeat in 36 years (cf. Section 7.4). Accordingly, despite both solutions show appropriate features for their adoption in MBPTA, the LFSR PRNG is taken as a reference in next sections.

7.2 PRNG for Multicore Platforms

As described in previous section, the reference LFSR PRNG generates a 32-bit number on every invocation. In spite of the quality of the generated numbers, the rate at which the PRNG can provide the required number of bits is of paramount importance. Based on the baseline LFSR PRNG, this section presents its integration in a multicore processor that meets MBPTA requirements and keeps costs low by sharing PRNGs across platform resources.

7.2.1 Multicore Requirements

The PRNG is integrated in a reference architecture based on the LEON3-MC previously introduced in Section 3.1.2. The reference architecture has N_c cores, each of which comprises private first level instruction and data caches (IL1 and DL1). Both caches are set-associative and use random placement (RP) and random-replacement (RR) (KAQ+13a) to meet with MBPTA time randomization properties. Each core has fully-associative data and instruction TLBs using RR. Cores are connected to a partitioned second level cache unified for data and instructions (UL2) (PQC+09) via a bus that uses random arbitration (JKA+14). The UL2 cache also employs RP and RR (KAQ+13b). UL2 cache misses are sent to main memory through a memory controller that has a request queue per core. Arbitration across cores for that request queue is random, like for the bus.

Time randomized resources that are accessed frequently, such as the first-level caches or the on-chip bus require random bits every few cycles. In particular, the RR policy and the random arbitration require the PRNG to generate several random bits on every invocation. The frequency at which a resource requires random bits determines whether they can be generated with a software PRNG or, alternatively, a hardware PRNG is required. Table 7.1 shows the frequency and the number of random bits required by the different time randomized resources.

Random Placement (RP) requires random bits once per run of the software unit for which a pWCET estimate has been derived. The reason behind this is that the mapping of addresses to sets, which is determined by RP, is fixed during the execution of a program and changes only across runs. Given that random bits for RP are needed at such coarse granularity, they can be provided by software means, e.g., by the OS or any other system software writing the random bits into a special register in the architecture. The hash function used to implement RP requires 32 bits. Note that each core will use its own set of random bits for RP in the UL2 so that the placement used by each core is independent and one core can change its random bits whenever needed.

Random Replacement (RR), the access to the bus, and the memory controller instead, all require random bits at much higher frequency. RR requires random bits

Table 7.1: Random bit requirements in reference multicore and LEON3-MC.

Resource	Frequency of Use	Worst Timing Requirements	Level	Bits Required	LEON3-MC Bits
IL1 RP	per run	SW dependent	SW	32	32
DL1 RP	per run	SW dependent	SW	32	32
UL2 RP	per run	SW dependent	SW	32	32
IL1 RR	per miss	Every cycle	HW	$\log_2(N_{ways}^{IL1})$	2
DL1 RR	per miss	Every cycle	HW	$\log_2(N_{ways}^{DL1})$	2
UL2 RR	per miss	Every cycle	HW	$\log_2(N_{ways}^{UL2}/N_c) \times N_c$	4
ITLB RR	per miss	Every cycle	HW	$\log_2(N_{ways}^{ITLB})$	6
DTLB RR	per miss	Every cycle	HW	$\log_2(N_{ways}^{DTLB})$	6
Bus	per access	Every cycle	HW	$\log_2(N_{cont}^{bus})$	2
Memory controller	per access	Every cycle	HW	$\log_2(N_{cont}^{mem})$	2

on every miss in the corresponding cache and Translation Lookaside Buffer (TLB) as, in the event of a miss, a victim in the corresponding set has to be randomly selected. Thus, the number of bits required in every case equals \log_2 the number of ways of the cache and TLB entries. As UL2 is partitioned across cores, each core needs its own random bits for RR in its own UL2 cache ways. For instance, assuming the LEON3-MC architecture with an 8-way UL2 cache and $N_c = 4$ so that each core has 2 ways in UL2, 1 random bit would be required per core to choose among its 2 ways, thus 4 random bits in total for the UL2 cache. Finally, the bus and the memory controller require $\log_2(N_c)$ bits for every round of arbitration, so 2 bits each. Consecutive accesses can occur in caches, bus and memory controller, so up to 72 random bits may be required at every cycle in the worst case in the multicore. The count of 72 bits includes 17 bits per core, 2 for bus arbitration and 2 for memory arbitration (cf. last column of Table 7.1).

7.2.2 Sharing PRNG Modules

In principle, a single 32-bit number generated by the PRNG can be shared among several resources. For instance, in the LEON3-MC that $N_c = 4$ and all DL1 and IL1 have 4 ways, TLBs have 64 entries and the UL2 has 8 ways; the number of bits required are 2 in the case of DL1 and IL1, 6 for DTLB and ITLB, and 4 for the UL2 (1 per core) (cf. last column of Table 7.1). This totals 17 bits per core. Hence, in theory, a 32-bit random number generated by a PRNG can be shared among the different resources of one core, reducing the need for PRNG devices.

7.3 SIL 3 Compliant PRNG

However, while the PRNG has been proven random by testing the produced bit sequences conveniently (cf. Section 7.1), it must be demonstrated that sharing a PRNG does not break randomness by creating some type of undesired correlation. For instance, first level caches require random bits much more frequently than the UL2. This may affect the random bits UL2 is given every time, which could create some type of correlation. In order to avoid those correlations by construction we adhere to the following two design principles:

Principle 1. Each core has its own PRNG, which is shared across all its cache memories (DL1, IL1, DTLB, ITLB, UL2 partition), see Figure 7.3a. Each component uses a fixed subset of the bits as depicted in Figure 7.3b (e.g., DL1 bits 0-1, IL1 bits 2-3, DTLB bits 4-9, ITLB bits 10-15, UL2 partition bit 16). In order to avoid any correlation between the events in the different components, a new random number is produced every cycle. Thus, the bitstream obtained by each component is completely independent from all other components and its randomness can be tested in isolation (RSN+10). For instance, IL1 uses the bits in positions 2, 3, 34, 35, 66, 67, etc., which are also random as shown later in the evaluation section. As an alternative, one could buffer some random bits (e.g., 64) and let caches consume them so that whenever there are less than 34 (enough to provide 17 bits in current cycle and 17 in next cycle), a new 32-bit random number is produced. This saves power by producing fewer random numbers, but would make it very hard to test the bitstream received by each cache as it cannot be determined a priori.

Principle 2. Randomly arbitrated shared components (e.g., the bus and the memory controller) need a PRNG producing random bits every cycle. Producing the random numbers on-demand when a task requests access to a shared resource is not convenient because it makes the bitstream observed by a task on that component depend on other activities of tasks in the other cores, which are unknown at analysis time. Those components can use idle bits from one of the cores (e.g., bits 17-18 and 19-20 from core 1 for the bus and the memory controller respectively) as illustrated in Figure 7.3b. The use of a separate set of bits and producing random numbers every cycle prevents correlations as for non-shared components.

Following these two design principles, in the reference processor architecture four PRNG modules – one per core – are employed as depicted in Figure 7.3.

7.3 SIL 3 Compliant PRNG

PRNG's malfunction invalidates the pWCET estimates derived for safety-related software. In particular, if the PRNG fails to deliver a new random number on each processor cycle it compromises the correctness of time randomized resources and hence, the soundness of the pWCET estimates. As a consequence, when MBPTA is

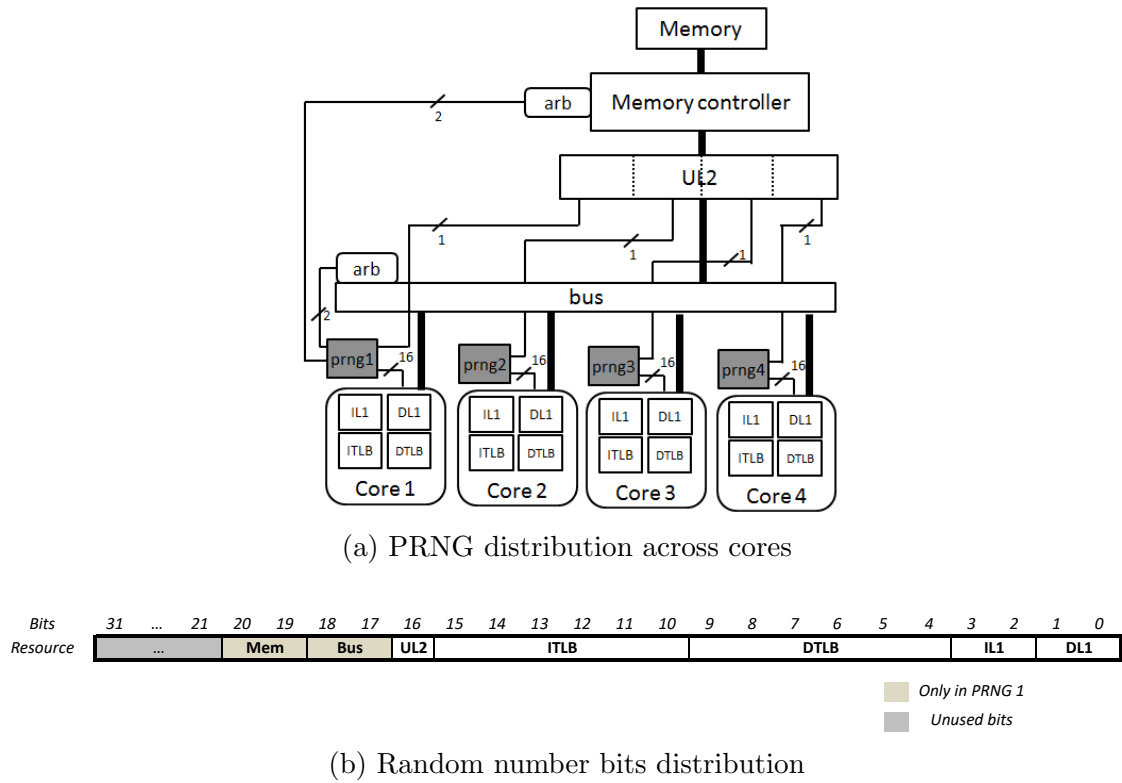


Figure 7.3: PRNG integration in the LEON3-MC architecture.

used for instance as a baseline to dimension a static cyclic scheduler in accordance to IEC 61508-3 Annex F.5, the resulting software executable may contain design faults derived from PRNG's malfunction and can result in system level failures (e.g., deadline overrun). To increase the level of reliance of MBPTA, this section presents a certification cognizant PRNG design to achieve IEC 61508 compliance. This is important as MBPTA, and hence the proposed PRNG design, has limited evidence of its successful use in safety critical environments due to its novelty. In addition, IEC 61508-3 requirements for support tools require the identification and mitigation of potential failures in tools that may affect the executable software (IEC 61508-3 §7.4.4.5).

The PRNG is part of the timing analysis tool and has a direct impact on the runtime timing behavior of processor resources such as caches that depend on the outcomes of the PRNG for the placement and replacement policies. In this section we apply IEC 61508 SIL 3 level safety related design principles to it. SIL 3 is the highest integrity level that can be claimed for single-chip implementations, as higher levels (i.e, SIL 4) require off-line redundancy according to IEC 61508-2 Annex E.

7.3.1 Safety Requirements

A failure in the PRNG is considered dangerous if it impairs timing correctness. For instance, a wrong number sequence can starve one or some cores (e.g., by never granting memory access to a core). To avoid this kind of situations, we define the following safety requirements (REQ_SR) for designing the PRNG:

REQ_SR_1 (Safety Function). *The PRNG shall provide a new random number on each processor cycle.*

REQ_SR_2 (SIL). *The PRNG is developed according to SIL 3 design principles.*

REQ_SR_3 (HFT). *The HFT of the PRNG is 0, i.e., a single fault leads to the loss of the safety function.*

REQ_SR_4 (DC). *The PRNG implements safety measures with a Diagnostic Coverage of $DC \geq 90\%$.*

REQ_SR_5 (Error reaction). *Detected internal errors trigger a system reset and lead to the safe state.*

7.3.2 Safety Techniques

Safety requirements 1 to 5 are fulfilled by a set of safety measures and techniques that (i) reduces the probability of systematic faults and (ii) controls faults during operation of the PRNG (Table 7.2).

Systematic (STM) faults are mitigated by using a compliant FSM, thus reducing the probability of introducing systematic faults throughout the PRNG development process (STM_ST_1 in Table 7.2). The development of the PRNG in a Field-Programmable Gate Array (FPGA) includes the techniques and measures listed in IEC 61508-2 Table F.2 to avoid introducing faults during ASIC design and development.

The control of random (RND) faults instead, is done during operation by the architecture of Figure 7.4. This architecture includes runtime safety diagnostic mechanisms (RND_ST_2 and RND_ST_3) independent from the particular PRNG implementation. The first runtime diagnosis technique, Monitored Redundancy (RND_ST_2 in Table 7.2), is achieved by deploying two replicated PRNGs operating in lockstep and by comparing their outputs with a two-out-of-two (2oo2) voting element as depicted in Figure 7.4. This means that both PRNGs must operate in coordination all the time. Any fault causing a discrepancy in their outputs will be detected and reported as an error by the voter. In systems that already include off-chip redundancy to conduct their safety related functionality, measure

Table 7.2: Safety measures and techniques in the PRNG.

ID	Technique	IEC 61508	DC	Description
STM.ST_1	Compliant FSM to reduce the probability of systematic faults	Table F.2 (part 2)	n/a	Techniques and measures on the FPGA development phase: 1) Design description 2) Synthesis 3) Test 4) Placement, routing and layout generation
RND.ST_2	Monitored redundancy	§A.2.5 (part 7)	High	2 PRNGs with 2oo2 voter comparison
RND.ST_3	Temporal and logical monitoring	§A.9.4 (part 7)	High	Watchdog timer for behavior and time monitoring
CC.ST_4	Physical separation	§E.1 f) (part 2)	n/a	Avoid short circuits and crosstalk
CC.ST_5	Sufficient design independence between elements	§7.4.3.4 (part 2)	n/a	Diversity between the two channels

RND.ST_2 can also be implemented off-chip, with each of the independent channels integrating a single PRNG and comparing the sequences of random variables with channel crosscheck.

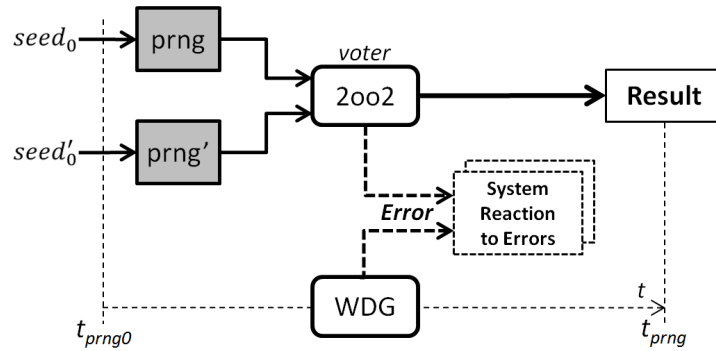


Figure 7.4: Safety architecture for the PRNG modules.

Moreover, to detect anomalies in the computation of random numbers, a watchdog timer (WDG) is incorporated (RND.ST_3 in Table 7.2). The WDG is configured with a time deadline that shall never be reached. When the two PRNG are fed with a new seed (t_{PRNG0}) the WDG timer starts a countdown and is re-triggered at the time at which the result is obtained (t_{PRNG}) provided that the computed number is different to that obtained in the previous cycle. Whenever

the WDG reaches its deadline, an error is reported. This guarantees that a new random number is updated at every cycle, detecting unexpected delays in their generation or voting errors. Given that both PRNGs need to coordinate in computing the same random number sequence, they need to be initialized with the same seed at the same time so that $seed_0$ equals $seed'_0$ in Figure 7.4.

In order to prevent two identical faults from affecting both PRNGs simultaneously, design measures to reduce the probability of common-cause (CC) failures shall also be contemplated. Common-cause failures can originate from external interference, systematic design failures, lack of spatial separation, etc. Measures to reduce their probability include maximizing separation and independence between both PRNG to avoid short circuit and cross talk between both channels (CC_ST_4) and applying diversity by using, for example, different types of logic and/or gates in each PRNG, or by designing different bit cells to implement seed registers (CC_ST_5).

7.3.3 System Reaction to Errors

Table 7.3 lists a set of possible errors and the defined system reactions in the safety architecture of Figure 7.4. As explained above, any fault resulting in dissimilar outputs of both PRNGs (SR1 and SR2 in Table 7.3) is reported by the 2oo2 voting element. However, if both PRNGs do not update their value in the same cycle (SR3 in Table 7.3) (for example owing to simultaneous failure in both seed registers) the voter may not notice it, leaving the diagnosis responsibility to the watchdog. The same happens whenever a delay occurs in both PRNGs or in the voter, thus causing the absence of the random number when required (SR4 in Table 7.3).

Different system reactions are possible. For this design, a system reset is triggered in all the defined situations as specified in requirement REQ_SR.5. During reset and system initialization, the system must remain in the safe state as described in IEC 61508. Then, health check for the PRNG is carried out at start-up and the system only starts normal operation if the check-up step is successfully passed.

Table 7.3: System reaction to errors.

ID	Error	System Reaction
SR1	Incorrect value in one PRNG	Detected by 2oo2 voter. Trigger a reset.
SR2	Missing value in one PRNG	Detected by 2oo2 voter. Trigger a reset.
SR3	Number repetition in the result	Detected by WDG timer. Trigger a reset
SR4	Missing value in the result	Detected by WDG timer. Trigger a reset

7.4 Evaluation

This section evaluates the PRNG properties required for MBPTA compliance and described in Section 7.1.1. We assess the properties of the PRNG based on its integration in the LEON3-MC quad-core prototype.

7.4.1 LFSR's Sought Properties

The properties of the LFSR PRNG are evaluated through specific tests that statistically prove that bit sequences produced by the PRNG are truly random and have sufficient period (RSN+10). Those tests look for a wide variety of repetitions, patterns, imbalance in the values, among other properties. In addition, other non-functional aspects like area, energy and delay are evaluated.

Randomness

We use the test battery provided by the US National Institute of Standards and Technology (RSN+10) to assess the randomness properties of LFSR. LFSR passed 187 out of the 188 tests for a set of 100 sequences of 400,000 bits each¹. Only the *Linear Complexity* test was failed. This test is devised to determine whether random bits have been produced by a LFSR. In particular, the test computes whether one bit is produced as the combination of several others, which is the case for a LFSR where each bit is produced by XNORing several others as shown in Figure 7.2(a). Thus, although the LFSR produces highly random sequences it cannot pass this test. Interestingly, this weakness is also an advantage in that it simplifies hardware implementation.

In addition, we have further validated that the subset of bits that each component would receive (e.g., bits 2-3 for the IL1) also passes the tests (RSN+10). Results show that all subsets passed *all* tests (including the one failed by the whole sequence). This occurs because any arbitrary subset of a random sequence is still random and the sub-sequences obtained by selecting some of the bits break the dependence that made one test to be failed for the original sequence.

Period

The period of the LFSR is particularly large and it was the main differentiating factor over the MWC PRNG. Given a LFSR of R bits producing 32-bits per cycle, it would repeat the sequence after $2^R/32 - 1$ cycles. In other words, assuming

¹Note that the authors of the tests recommend using no less than 55 sequences of at least 390,000 bits each.

7.4 Evaluation

$R = 168$, a processor operating at 1GHz, and the need to deliver 1 random number per cycle, the random number sequence only repeats in 3.7×10^{32} years.

Area, Energy and Delay

The implementation overhead of the proposed LFSR is estimated using the CACTI tool (MBJ09), which is an accurate delay, energy and area model for cache memories. The LFSR delay is very small: it fits in one cycle even for very high operating frequencies (below 0.1ns in 65nm technology). Its energy is also very low: 2-3 orders of magnitude lower than the energy required for a cache read operation (0.08pJ per random number versus 24pJ per read access for a 8KB 4-way 16-byte line cache). Moreover, a single LFSR suffices to provide random bits to all components in a core so its energy consumption is negligible even when it is replicated for redundancy as shown in Figure 7.4. Therefore, the LFSR energy consumption is negligible even if it needs to be replicated to be SIL 3 compliant, as shown in Figure 7.4. Note that the overhead is very low given that a random number is enough to feed all components in a core.

7.4.2 PRNG Integration in a Multicore Prototype

The proposed LFSR is integrated in the LEON3-MC. All LFSR modules required by the multicore (cf. Figure 7.3a) to fulfill MBPTA requirements have been integrated in a single hardware module (*randbank*) that is attached to the advanced peripheral bus (APB) of the SoC. The *randbank* module has a pool of 32-bit registers devoted to the initialization of the different seeds required to randomize multicore elements as described in Section 7.2. There are two types of seeds: random and fixed. Random seeds provide processor features with bits that change at every cycle after initialization. Fixed seeds keep the same value until it is modified by overwriting the contents of the register. Therefore, random seeds are used by the features gathered in Table 7.1 that require random bits every cycle (e.g., RR policies of caches and bus arbitration). Fixed seeds can be used for the cache placement as the same cache layout must be kept for a given execution and it thus requires random bits only once per run. As noted in Section 7.2, the random bits for RP can be provided by software means given the required coarse granularity. Consequently, in the experiments reported here we analyze the RR policy of the multicore prototype to evaluate the proposed LFSR PRNG and thus, random seeds are used. The *randbank* seed initialization is carried out writing in a specific address of the APB address space where this component is mapped to.

To evaluate *randbank* integration the LFSR random bits are routed to the L1 caches to carry out the replacement in the data and instruction caches of the LEON 3 processor. The EEMBC Autobench benchmark suite (Poo07), a well-

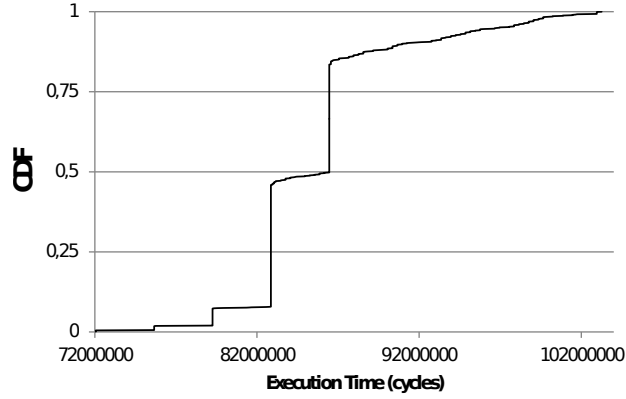


Figure 7.5: CDF for 1000 runs of the matrix benchmark.

known suite reflecting the current real-world demand of some automotive embedded systems, was run 1,000 times with the RR policy, collecting execution time results for each run. Figure 7.5 shows the empirical cumulative distributed function (CDF) of the observed execution times when using the RR policy and the random bits generated by the LFSR. These experiments show the good properties of the developed LFSR as the collected execution time values were proven independent and identically distributed (i.i.d.) in all benchmarks, following the i.i.d. tests described in (CSH+12). In contrast, the measurements collected with the default random policy in the LEON 3 processor failed those tests. In Figure 7.5 steps are caused by different (random) placements that may lead different addresses to compete (or not) for the space in some particular cache sets during the whole program execution. Variations within each step show smooth distributions caused by the RR policy using the LFSR, as expected from a good PRNG, which properly randomizes each replacement keeping events independent and identically distributed.

7.5 Summary

The confidence of MBPTA results rest on its ability to derive trustworthy WCET bounds. To this end, it is crucial to ensure that the platform is MBPTA-compliant and that hence, it includes time randomization properties. The SoR is a key component to ensure the platform level randomization needed by MBPTA. In this chapter, we have described the benefits of a hardware PRNG for its use as an SoR in MBPTA-compliant platforms. Based on a reference LFSR PRNG, we have designed an IEC 61508 compliant PRNG for multicore processors that meets with the desirable properties for its integration in MBPTA-compliant platforms.

Results show that the LFSR PRNG produces high-quality random numbers

7.5 Summary

with sufficiently long period. In addition, the bits produced by the LFSR are shared among different platform components to increase the efficiency and reduce the number of required PRNG modules. The SIL 3 PRNG implementation includes safety design procedures and techniques that improve the level of reliance of the MBPTA approach. In this way, we guarantee that platform random timing properties for which the pWCET is computed are preserved at runtime and we reduce the likelihood of introducing faults in the system by the use of MBPTA. Therefore, the work in this chapter paves the way towards the adoption of MBPTA in the development and certification of safety related and mixed-criticality systems in multicores.

Chapter 8

Practical CAST-32A Compliance on COTS Multicores

COTS multicore processors are the preferred industry choice to deal with the increasing integration demands in the embedded domain. The use of readily available and tested COTS components considerably shorten the development time and associated costs. In spite of these benefits, however, most COTS processors are primarily designed towards increasing the average performance at the expense of increasing timing unpredictability. In addition, the often incomplete – rather functional – platform documentation, seriously limits providing guarantees on timing and certification as safety standards do not provide sufficient guidance yet for such architectures. The latter question has motivated the joint endeavor of certification authorities in the airborne domain to provide support for the certification of systems integrating multicore processors. As a result, they have published the CAST-32A ([CAST-32A](#)) support document, which is, at the time of writing, the only explicit certification guidance for multicore architectures.

In this chapter, we focus on assessing the practical compliance of multicores with respect to certification guidance. In particular, we provide a deep analysis of CAST-32A objectives and discuss the feasibility of achieving them with current COTS multicore design trends. We show potential limitations that embedded system designers may face in achieving those goals. We then evaluate CAST-32A principles on a specific COTS multicore processor, the P4080 ([P4080](#)). We show that, despite CAST-32A is a step forward for software planning, development and verification on multicore systems, the application of its objectives is not straightforward and requires in-depth analysis of the architecture and appropriate hardware support to deal with interference channels.

The rest of this chapter is structured as follows. Section [8.1](#) introduces the definitions and main objectives of CAST-32A position paper. Section [8.2](#) provides the analysis and feasibility of achieving those objectives with current COTS design

trends. Section 8.3 evaluates CAST-32A objectives on the P4080 processor, together with some quantitative results that support the claims made in this chapter. Finally, Section 8.4 presents the main conclusions of this work.

8.1 Overview of CAST-32A Principles

CAST-32A analyzes multicore processor elements potentially impacting the safety, performance and integrity of a software airborne system. It is partially motivated by the fact that current standards in civil avionics cover single-core systems, since those standards precede multicore usage in civil avionics.

8.1.1 General Definitions

CAST-32A builds on several definitions that classify multicores based on the type of support they provide to reach its goals. Emphasis is put on hardware/software elements that create ‘coupling’ among software tasks running on the multicore, and hence may lead to interference among them. Below, we introduce those CAST-32A definitions related to timing.

CAST-32A Definition 1 (Robust Resource Partitioning (RRP)). *RRP requires (i) software partitions not to manipulate the code, I/O or data of other software partitions; (ii) software partitions not to consume more than their assigned portion of shared resources; and (iii) hardware failures unique to a software partition not to affect other software partitions.*

CAST-32A Definition 2 (Robust Time Partitioning (RTP)). *RTP requires identifying and mitigating inter-partition interferences such that no software partition exceeds its deadline even in the presence of software executing simultaneously in other cores.*

CAST-32A Definition 3 (Robust Partitioning). *Robust partitioning encompasses both, RRP and RTP. CAST-32A classifies multicore processors into two categories depending on whether or not they provide robust partitioning.*

CAST-32A Definition 4 (Interference Channel). *An interference channel refers to a platform property that may cause interference between independent applications (i.e., with no explicit data or control flow between them).*

CAST-32A Definition 5 (Configuration Settings). *The configuration settings of the multicore cover any software-configurable element that affects timing behavior, e.g., frequency and cache partitioning. Special care is required to prevent the inadvertent changes of those settings that affect planned application’s timing behavior.*

8.1 Overview of CAST-32A Principles

8.1.2 CAST-32A Objectives

Table 8.1 lists CAST-32A objectives related to interference analysis and WCET determination. Their identifiers refer to whether the objective relates to planning (PL), resource usage (RU), software verification (SWV) or error handling (EH).

Table 8.1: Summary of CAST-32A objectives ([CAST-32A](#)).

Phase	ID	Description
Software Planning	PL_1	Include multicore specific planning details in the SW plan doc. <i>Specific processor, number of cores, SW architecture, dynamic SW features, IMA-like systems, Robust Partitioning methods and tools for development and verification.</i>
Planning and Setting Resources	RU_1	Determine configuration settings that enable to satisfy the functional, performance and timing requirements.
	RU_2	Critical configuration settings shall be static and protected against unintended modifications.
	PL_2	Include a high level description of shared resource usage and dynamic HW features in the HW and SW planning docs. <i>Intended shared resource allocation and verification to prevent resource capabilities from being exceeded.</i>
Intf. channels and Resource usage	RU_3	Identify interference channels and verify the chosen means of mitigation. <i>Interferences caused by shared memory, shared cache, interconnect, shared I/O or any other shared resource.</i>
	RU_4	Identify available resources in the final configuration, allocate them to the applications and verify that the demands do not exceed the available resources. <i>Consider worst-case resource usage scenarios.</i>
SW Verification	SWV_1	Verify that all software components function correctly and have sufficient time when all the software is executing in the intended final configuration. <i>Depends on the platform classification:</i> <i>1. Platforms with Robust Partitioning: SW verification and WCET analysis can be done separately for each SW app.</i> <i>2. All Other Platforms: If interference is mitigated, the verification of such components can be done separately. Otherwise, verification and WCET analysis shall be done with all software components executing together.</i>
	SWV_2	Data and control coupling between SW components is exercised during testing (for inter-partition data and control flows on different cores).
Error Handling	EH_1	Planify, design, implement and verify means to detect and handle failures in a fail-safe manner (e.g. safety-net external to the multicore).

The objectives of Table 8.1 can be grouped into four high level principles:

1. **Determining the final configuration.** The designer shall determine which is the intended final configuration that will enable to satisfy system requirements (*RU_1*). This configuration is protected against unintended modification at runtime (*RU_2*). The decisions taken at this phase should be documented as required in objective *PL_1*.
2. **Managing interference channels.** It is required to identify interference channels in the intended final configuration and to define the means to either avoid interference by design or upper-bound it so that timing deadlines are not exceeded (*RU_3*). Upper-bounding interference involves analyzing the use of shared resources and designing the means to control multicore contention. This should be documented in the appropriate certification deliverables (*PL_2*).
3. **Verifying the use of shared resources.** Resource usage and data and control flows among cores shall be verified by guaranteeing that in the chosen final configuration the software does not exceed the use of available resources even in worst-case scenarios (*RU_4*, *SWV_1* and *SWV_2*).
4. **Error Management.** The system shall include features for multicore specific error detection and handling. CAST-32A suggests using an external *Safety Net* for this purpose.

8.2 Interpretation and Achievability of CAST-32A Objectives

The safety argumentation of CAST-32A is based along the partitioning line of reasoning: guaranteeing robust partitioning is crucial at the time of meeting the objectives of CAST-32A, specially for allowing incremental verification of different software components integrated in the multicore system as stated in objective *SWV_1*. This section provides a critical analysis of CAST-32A objectives, their achievability under current design practices.

8.2.1 Robust Resource Partitioning (RRP)

CAST-32A definition of RRP translates into i) spatial partitioning, ii) resource quota monitoring and enforcement, and iii) fault containment at (software) partition level.

Spatial Partitioning

The definition of spatial partitioning in CAST-32A is analogous to spatial independence requirement in IEC 61508 where the data of a software element shall not be modified by another element or partition. This can be achieved with MMU/MPU support in most current COTS multicore architectures.

Quota Monitoring and Enforcement

The second aspect of resource partitioning is to ensure that software partitions do not consume more than their allocations of shared resources. Achieving this requirement with current COTS multicores involves many difficulties and it depends on the way in which partitioning is implemented and enforced in the shared resources. For some shared resources, like a data array in memory, partitioning can be statically defined and enforced in hardware as defined for achieving spatial partitioning. However, the contention in *bandwidth resources*, like buses, may be influenced by many factors (e.g., request overlap) and partitioning can only be implemented by monitoring and enforcing some events (like maximum number of requests) at runtime. For each shared resource it should be identified which events provide more accurate information about resource usage.

In terms of quota monitoring, intuitively, while access counts can provide a good approximation, different type of accesses use the shared resource for different time. This does not just require deriving this information from available manuals, but also checking whether the target platform provides support for monitoring architectural events (e.g. PMCs) that allow tracking the desired type of accesses. For instance, in the LEON4 architecture, the last-level (L2) cache is write-back. L2 cache misses not evicting dirty data take shorter than those evicting dirty data. However, the L2 miss count PMC only captures non-dirty misses (JFA+16). While authors (JFA+16) manage to upper-bound the number of dirty misses with existing counters, for other scenarios PMCs cannot provide enough support for access tracking, hindering a tight quota monitoring.

Furthermore, some shared resources, like AMBA – one of the most used interfaces for communication – buses, allow a single request to hold the bus for an unbounded duration (JAQ+14). This poses new challenges, since additional means are required to determine whether hardware masters (i.e. resources allowed to start transactions on the bus) use this unbounded-duration feature for requests. This, however, requires deep hardware understanding, which might not be possessed by end users. Moreover, PMCs counting access types might not suffice. Instead, PMCs counting the time each core uses the shared resource (hence potentially delaying the other cores) may also be required.

The difficulty of implementing quota enforcement depends on the target re-

source and the hardware support provided. We classify shared resources as space and bandwidth resources.

First, for *space resources*, like the data array in a cache, some processors provide partitioning so that one task cannot evict the data of another. However, despite cache space partitioning, some tasks can still impact others. This may happen if partitioning is not implemented at the cache bank level. That is, tasks share the same bank though the cache ways in that bank are split among tasks. This may make cache accesses of one task delay the accesses of another task (PQC+09). Further, caches may have queues to hold cache miss requests, which, if shared, can create huge contention among tasks despite cache ways are split (VYF16). This extremely low-level type of information about hardware requires the involvement of a hardware expert with proper access to manuals and also making experiments to empirically determine whether this situation can happen (analyzed in the experimental section).

And second, *bandwidth resources*, can be split in time and space. For instance, a task can be given 50% of the bus bandwidth over a period of 10,000 cycles. This can be implemented with alternative full-access/no-access periods of 1,000 cycles, 2,500 cycles, 5,000 cycles, etc. Each of these ways to implement partitioning affects tasks' execution time. Further, given task τ_a and τ_b , where task τ_b makes a given number of accesses, how those overlap with τ_a 's requests has an impact on τ_a 's execution time. Moreover, request overlap can change drastically from run to run with different or even the same inputs. In practice, determining how tasks' requests overlap during operation is unaffordable (in the final configuration), so existing approaches build on the assumption that tasks overlap their requests in the worst possible manner (FJAQ+15; NPB+14). Worst overlapping occurs when each request of the task under analysis has the lowest arbitration priority and it becomes ready when all other tasks in other cores have pending requests.

Fault containment at the resource partition

For physical faults, any multicore shall be considered to form a single fault containment region, that is, it is not possible to guarantee that the immediate effects of any possible fault are limited to a single software partition in a multicore unless it is specifically addressed in the design (kop06). This occurs because, as a consequence of sharing the same silicon die, a number of shared elements become a single point of failure for all partitions simultaneously, such as the power supply or clock source. Thus, it cannot be claimed that different software partitions will fail *always* independently. CAST-32A suggests using an external safety net to enforce such *physical fault containment* at multicore platform level preventing propagation to other components.

Furthermore, for achieving RRP, CAST-32A requires that failures of hardware

8.2 Interpretation and Achievability of CAST-32A Objectives

unique to a software partition do not propagate to other software partitions. While partitioning techniques (e.g. hypervisor) have been commonly used to avoid the propagation of design faults (e.g. deadline misses) among software partitions (i.e. *design fault containment*), they do not usually account for hardware faults. Obviously, a failure in a time-shared (e.g. bandwidth) resource extends to all software partitions using it. For space resources, hardware partitioning techniques provide some degree of isolation but (physical) fault containment at the resource-partition level is challenging, which is better illustrated taking a way-partitioned shared cache as example. While it is possible to prevent a fault in a cache bitcell to propagate to partitions not using that bitcell, some hardware resources, such as sense amplifiers and data output buffers (used to read/write the data in cache), may be independent or silently shared, thus making a permanent or intermittent fault affect all software partitions in the latter case. Therefore, it is hard to draw the line between hardware unique to a software partition and the features shared on the underlying multicore platform. As a consequence, in general, end users *do not have the means* to sustain or validate this claim. Hence, the only way it can be deployed is that it comes provided by the hardware vendor.

In this regard, the requirements in IEC 61508-2 Annex E for on-chip redundancy are representative of the conservative restrictions needed to prevent *fault propagation* in a single chip. These requirements include, among others, a full common cause analysis that considers all possible faults that can affect to various channels and therefore result in the loss of physical separation. The common cause analysis shall consider the design, fabrication, construction, procedural and environmental factors and respective prevention measures are required, like controlling the effects of increasing temperature in various channels, using separate physical substratum blocks with a minimum distance among channels and assigning separate input/outputs for each channel (e.g., separate clock signals and power supply). However, this possibility of having hardware faults that may affect to more than one partition does not prevent from developing and certifying each software partition independently according to its assigned criticality level. Unlike CAST-32A, this restriction in IEC 61508 only applies when the different partitions conform different channels of a redundant architecture implemented on the same chip to achieve $HFT > 0$.

8.2.2 Robust Time Partitioning (RTP)

From previous discussion, it follows that a multicore will usually have some form of interference channels due to the difficulties of implementing RRP. Note that a form of interference channels arise when tasks have interactions in the time domain.

In this scenario, the RTP goal of “*no software partition exceeds its deadline even in the presence of software executing simultaneously in other cores*”, trans-

lates into deriving contention bounds for a task that hold for any load that the contender tasks can put on the shared resources. That is, the increment in time a task is assumed to experience (in the WCET estimate) due to contention through that interference channel, upper-bounds the actual interference it can suffer in the presence of any task. This type of bounds, which are referred to as fully-time composable (FJAQ+15), can be very pessimistic. For instance, let assume that τ_a generates n_a accesses to a round-robin bus of a given type t_1 that uses the bus a single cycle. Further, assume that there are another type of accesses t_2 that holds the bus for 10 cycles. For τ_a access, the worst situation is that a second contender τ_b generates accesses of type t_2 and that arrive at the same time of τ_a requests but are served first. Under a fully-time composable scenario τ_a is assumed to suffer an execution time increase of $10 \times n_a \times T$ cycles. That is, each request is assumed to suffer a 10x increase in its access time due to contention.

To reduce this overhead, a *partially-time composable* approach (FJAQ+15) can be followed. This approach accounts for the worst overlap of the *actual* requests of the contenders. This requires to determine i) the number and type of requests to the shared resources of all tasks (NPB+14); and ii) the time each request type holds the shared resource (FJAQ+15). The WCET estimate for a task is derived under a given *template* that describes the number and type of accesses performed by its contenders. That WCET estimate holds under any workload for which the actual number and type of requests performed by the contenders is smaller or equal to that assumed in the template. Coming back to the previous example, τ_a can be derived a WCET bound for a template that assumes K_1 access of type t_1 and K_2 of type t_2 . It is valid for any contender τ_b that fulfills $(n_b^{t_1} \times 1) + (n_b^{t_2} \times 10) \leq (K_1 \times 1) + (K_2 \times 10)$.

8.2.3 Individual Objectives

CAST-32A introduces new objectives in the different phases of the life-cycle to analyze and mitigate interferences, and control shared resource usage (i.e. limit multicore contention) in a given platform configuration. To that end, CAST-32A resource usage objectives (RU_1 , RU_2 , RU_3 and RU_4) impose new requirements in the design.

RU_1 is hard to achieve a priori in processors without RRP . This occurs because the impact of interference channels on software timing can be significant. Further, simply assuming the worst-contention (i.e. fully-time composable bounds) is in general impractical. In this situation, the configuration achieving software's timing requirements cannot be determined, until the final configuration is consolidated. With the partially-time composable approach (FJAQ+15), WCET estimates are derived under different templates and hence the work at integration reduces to check the template that upper-bounds contenders' request count (see example above).

8.3 Evaluation on Complex COTS Multicore

RU_2 in general can be achieved with proper hypervisor and/or OS support.

Regarding *RU_3*, some interference channels can be determined from processor manuals. Others, however, are not properly documented and hence, require an expert performance analyst that can create a set of microbenchmarks (FGQ+12) to determine whether further interference channels can exist.

RU_4 objective encompasses the successful achieving of those above, with their associated challenges.

Further, in terms of the general approach, it must be understood that objectives cannot be achieved in a sequential manner, as a first read could suggest. Instead, strong dependencies exist among them, that require considering all (or some) of them simultaneously.

8.3 Evaluation on Complex COTS Multicore

We select the P4080 (P4080) multicore processor as a representative of the complexity in modern COTS multicore architectures relevant for real-time industry (NPB+14; BGG+14). In this kind of COTS architectures the identification of interference channels and sources of execution time variability is challenging given the complexity of the architecture and limited information in manuals.

8.3.1 Defining Hardware Configuration Settings

We evaluate the first CAST-32A principle, *RU_1*, of selecting a suitable platform configuration by exploring different possible hardware setups (HWS) in the P4080 platform. Some hardware resources of the P4080 platform can be adjusted to applications' requirements by several configuration settings. Below, we list the most relevant customizable features that may influence performance, or partitioning, identified with a qualitative analysis of processor specifications (P4080; Bos13). A priori, it is not possible to determine whether a complex COTS multicore platform such as the P4080 can provide robust partitioning as defined in CAST-32A. CAST-32A requires identifying interferences in the intended final configuration. However, selecting a configuration involves determining a hardware partition setup, which in turn determines the interference channels.

Identifying Configuration Options

Configuration options can be classified as *core-local settings* and *shared resource settings*. The former provides the following options. The eight cores of the P4080 are logically independent, each core has its own boot and reset control. The user can select which specific cores to activate while others remain dormant. Similarly,

each core can be set to a different operation frequency. First level caches can be enabled or disabled. The second level backside cache can be either disabled or configured as data only, instruction only or unified (data and instruction) cache. Regarding the latter, *shared resource settings*, the following configuration settings are available:

- **CoreNet Interconnect:** The interconnect can be configured in up to 32 address windows (Local Access Windows, LAWs) to route transactions. Each LAW serves to define the internal connections by mapping different address regions to specific target devices (such as DDR controllers or peripheral interfaces). In addition, the configuration of each LAW includes a coherency subdomain identifier that allows to define which cores and caching elements should be maintained coherent.
- **CoreNet Platform Cache (CPC):** It supports flexible configurations on a per-way granularity. Each CPC has 32 ways (32KB per way) that can be configured as L3 cache or as static SRAM. For ways configured as L3 cache, cache partitioning is supported to assign each way to a logical partition. The ways configured as SRAM are mapped to a configurable physical address range. With the appropriate LAW and MMU configuration, the SRAM address space can also be segregated into different logical partitions.
- **Main Memory:** The two memory controllers have interleaving support. When enabled, interleaving can be configured to switch between memory controllers for every cache line transfer, every page line transfer or at every bank transfer. If disabled, the two memory controllers operate independently. Like for the SRAM, main memory can be divided into different MMU-restricted physical regions for each of the cores.
- **Peripheral devices:** I/O transactions can be controlled with the *PAMU* that acts as a MMU for I/O devices.
- **Interrupts:** They can be routed to any of the eight cores separately or to more than one core simultaneously.

Hardware Setup (HWS) Definition

Based on the configuration options listed above, we define several different HWS, see Figure 8.1. To that end, we establish a baseline configuration, common across all HWS, for the core local resources and we rather focus on the different possible setups for shared resources. To evaluate isolation, we narrow down to only one (safety-critical) application that needs to be strongly independent from the other

8.3 Evaluation on Complex COTS Multicore

seven and assume that each core hosts at most one software application. Taking this into account, in Figure 8.1, the resources used by the critical application (executed in c_0) are gray shadowed.

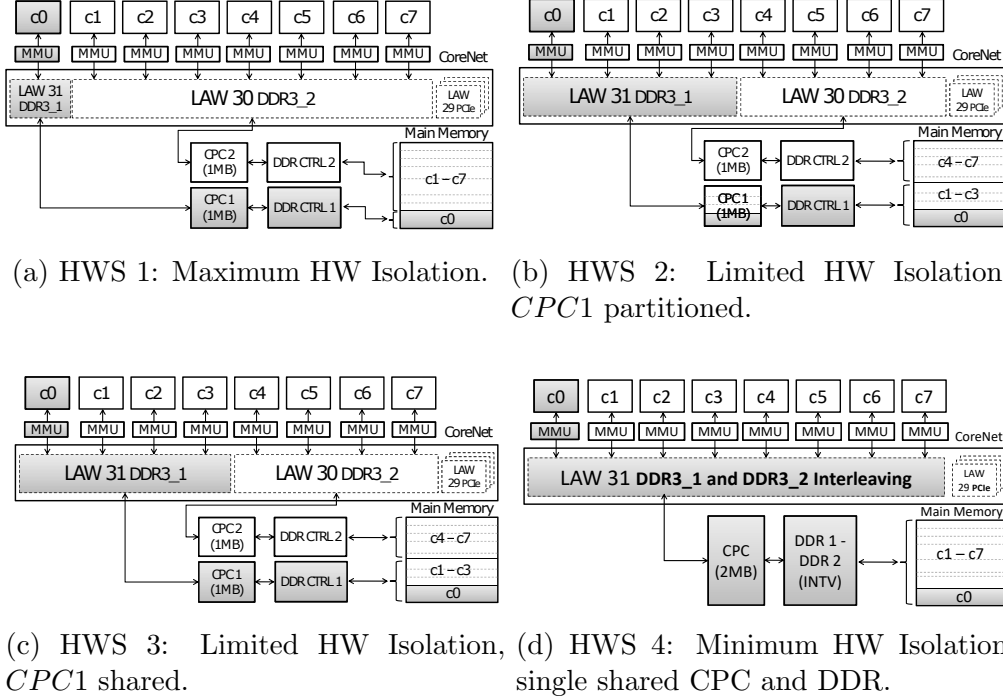


Figure 8.1: Examples of P4080 hardware setups.

Baseline configuration: All cores are activated with highest supported frequency and local caches enabled (DL1, IL1 and unified L2) for increasing performance. We focus on the memory hierarchy as shared resource and therefore we assume that the same peripheral is not shared among multiple cores.

(a) HWS 1: Maximum HW Isolation.

HWS 1 (Figure 8.1a) seeks to achieve the highest possible level of resource privatization (maximum partitioning) with the following configuration: (i) memory controller interleaving is disabled and $DDR3_1$ is exclusive for c_0 while $DDR3_2$ is shared among the remaining seven cores ($c_1 - c_7$); (ii) as each CPC corresponds to a memory controller, $CPC1$ is also restricted to c_0 and can be configured either as (private) SRAM, L3 cache or both. $CPC2$ may or may not be partitioned across the non-critical partitions. In particular, we use both CPCs in cache mode; and (iii) main memory is segregated in at least two MMU-protected memory regions, one for c_0 and

the rest for cores $c1 - c7$. If required, the latter can be further partitioned to have a separate address range for each core.

(b) HWS 2: Limited HW Isolation I.

This setup, shown in Figure 8.1b, provides better balance between resource privatization and sharing (and hence, efficiency): (i) memory controller interleaving is disabled and applications are distributed across the two memory controllers (e.g. *DDR3_1* for $c0 - c3$ and *DDR3_2* for $c4 - c7$); (ii) *CPC1* is partitioned by assigning one or more 32KB ways to $c0$. As in HWS 1, these ways may be setup as SRAM, L3 cache or both. The remaining ways of *CPC1* and *CPC2* may or may not be partitioned across the non-critical partitions. In particular, in *CPC1* we give 8 ways to $c0$ and 24 ways (shared) to $c1 - c3$; and (iii) main memory is segregated as in HWS 1.

(c) HWS 3: Limited HW Isolation II.

This setup matches *HWS 2*, but *CPC1* is not way partitioned but shared by $c0 - c3$, see Figure 8.1c.

(d) HWS 4: Minimum HW Isolation.

Most resources are shared across all cores for evaluating the impact of shared resources in the worst-case (Figure 8.1d): (i) memory controller interleaving is enabled so both memory controllers are interchangeably accessed by all cores; (ii) *CPC1* and *CPC2* are configured as L3 cache and shared across all cores; and (iii) main memory is segregated as in HWS 1.

8.3.2 Identifying Interference Channels

The timing behavior of an application is often affected by (hard-to-predict) jittery resources and interference channels that are difficult, if at all possible, to control. In COTS processors the impact of architectural features that affect on execution time is not usually known a priori and some features are poorly documented in processor manuals. Below we list some of the potential limitations and interference channels for each HWS derived from the analysis of the P4080. Note that other interference channels may exist. In fact, the challenge for the end user is to combine information in the manuals and previous experience to derive potential interference channels.

Note that the goal of this section is not to provide a characterization of the timing behavior of the P4080, which would be an extensive technical report. Instead, we provide the result of specific experiments providing key insights about achieving certification objectives. We show that even the most conservative partitioning setups can be subject to the impact of interference channels.

8.3 Evaluation on Complex COTS Multicore

Experimental setup

In this section we run simple microbenchmarks (*rsk*) stressing the memory hierarchy and interconnect. As introduced in Section 3.2, the *rsk* consists of a single loop with *load* or *store* instructions that traverse a data array of different sizes. In this section we define five *rsk*, *DL1h*, *L2h*, *L3h*, *L3m* and *mem*, where we access to a data array of 24KB, 114KB, 256KB and 768KB and 1MB (*ALLOC_SIZE*) respectively and a *stride* that ensures that a different cache line is accessed in each load or store operation. After traversing the array once, so that it is present in the caches, we execute 1000 iterations of the *rsk* and measure execution time and architectural events (PMCs). We use the PMCs available in each core of the P4080 to monitor processor cycles and memory related events (number of accesses, DL1 misses, L2 accesses, bus requests, etc.).

Results: Core local contenders

Each of the benchmarks is first executed in isolation to obtain a baseline latency for memory accesses (loads) at different levels of the memory hierarchy. Then, we execute same benchmarks concurrently with increasing number of *L2lh* contenders. Figure 8.2 depicts the results obtained for increasing number of contenders where the contender fits in L2 and hence it does not access to shared platform resources.

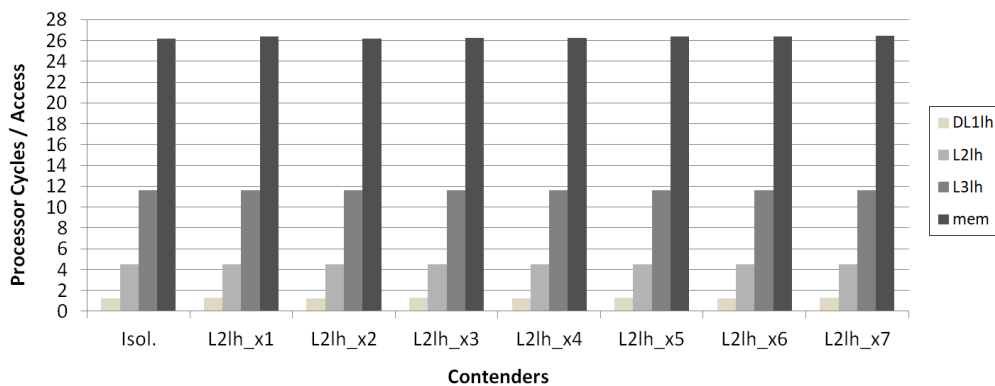
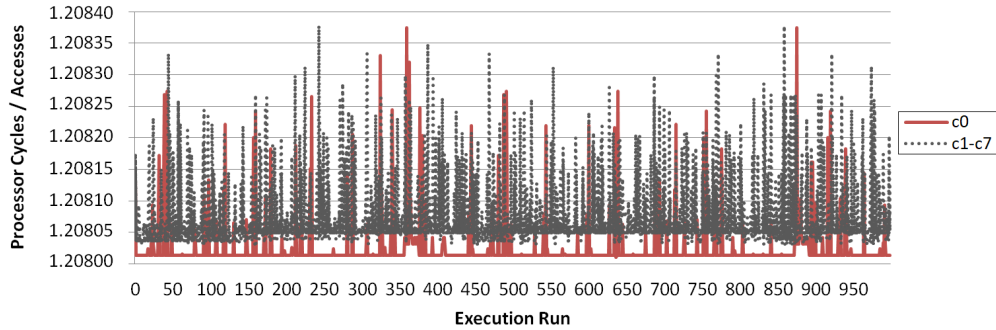


Figure 8.2: Impact of core local contenders on *rsk* *DL1lh*, *L2lh*, *L3lh* and *mem*.

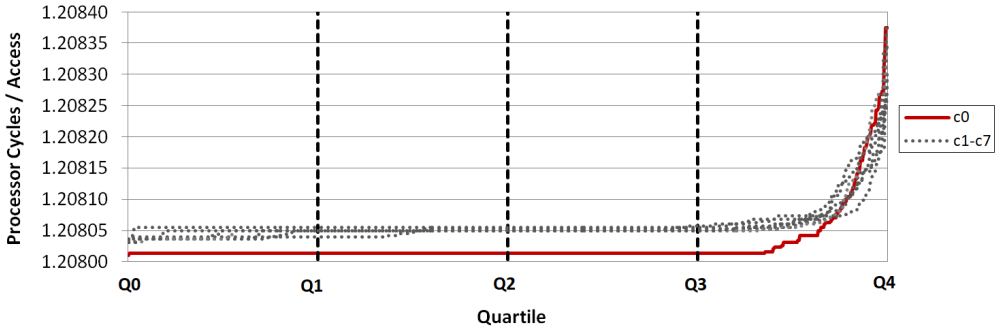
As expected, the results in isolation reflect how average memory access latency increases as going further in the memory hierarchy. At a first glance, the impact of core local contenders is negligible, between 0%-1% slowdown for *L2lh*, *L3lh* and *mem* *rsk* and between 0 and 5% for *DL1lh* *rsk*. With the aim of exposing the reason behind the 5% variation in *DL1lh*, below we report undisclosed sources of variability.

Heterogeneous behavior. We start by reporting some undocumented features impacting time. We execute *DL1lh* in each of the cores (ci) of the P4080 in isolation. We derive the DL1 hit latency on each core ci as depicted in Figure 8.3a across runs. The result of this simple experiment reveals that there are small *systematic* deviations from core to core. Our analysis of the data sample of 1000 runs on each core reveals that the central quantiles (Q1-Q3) of $c0$ (thus discarding outliers) are below the minimum value across *all* other cores (Figure 8.3b).

Even if the order of magnitude of this difference makes the variation negligible, 0.003% lower for $c0$ w.r.t. the other cores, its systematic nature reveals some heterogeneity in the physical design of the cores of the P4080 that is not reported in the (public) documentation. Thus, it is unknown whether the source of this difference may have noticeable (relative or absolute) impact in some programs.



(a) Observed execution times



(b) Quartiles

Figure 8.3: DL1 Hit latency variation across cores and runs.

Coherence. The coherence protocol is a potential interference channel. Apart from its impact in execution time for concurrent memory accesses (evaluated by Nowotsch et al. in (NP12)), our evaluation results show execution time variations even when one core is running *DL1lh* – hence only accessing local caches – in

8.3 Evaluation on Complex COTS Multicore

isolation (with the rest of the cores dormant). By analyzing the observed events, we conclude that those variations are related to snoop overheads: as depicted in Figure 8.4, the core under analysis receives unexpected snoop requests (monitored through PMCs), that correlate with execution time variation. As for core-to-core differences, execution time impact is negligible, but uncontrolled. Thus, an interference channel exists and its potential magnitude is unknown. This has also an impact on the results previously reported in Figure 8.2 where contenders in other cores cause a 0-5% execution time variation in *c0* running *DL1lh*. Despite the activity in the other cores does not access to shared resources, fact that is guaranteed by reading the access count to the bus interface unit, this experiment reveals that there is an influence in execution time.

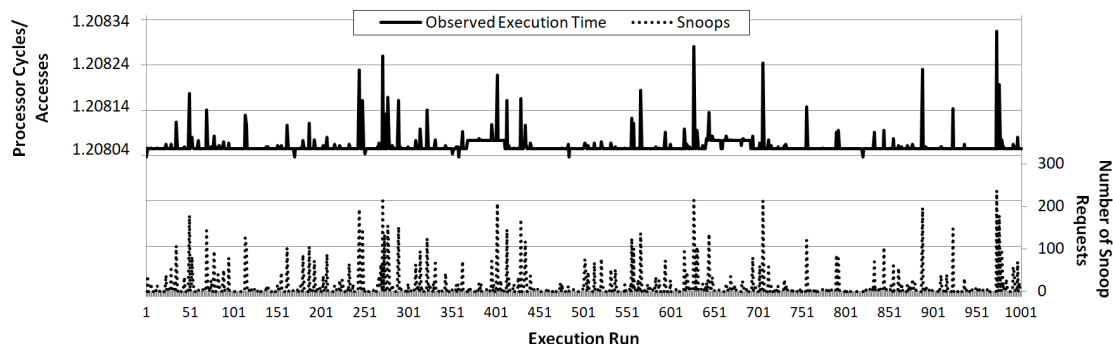


Figure 8.4: Impact of coherence on execution time in isolation.

Results: Shared Resources

Figure 8.5 compares the execution time of a microbenchmark performing sustained L3 (CPC) load hits (*L3lh*) by using 256KB of the 2MB *CPC* cache memories with increasing number of contenders (from 1 to 7) that perform all of them either sustained CPC hits (*L3lh* in Figure 8.5a) or CPC misses by using 768KB *rsk* (*L3lm* in Figure 8.5b). This experiment is performed in the four different HWS defined in Figure 8.1 and results are normalized with respect to the execution time observed in isolation.

As shown in Figure 8.5a, whenever all cores perform CPC hits, the influence that *c0* experiences from contenders in HWS 1 is small, with a slowdown below 1% with all cores running together. This slowdown is the combination of core-local sources of jitter and the shared use of the interconnect. The information about CoreNet is scarce, which prevents from concluding whether it is an interference channel since it may contain buffers or queues to hold pending requests. This would make it a statefull resource and hence a source of interference. When, in

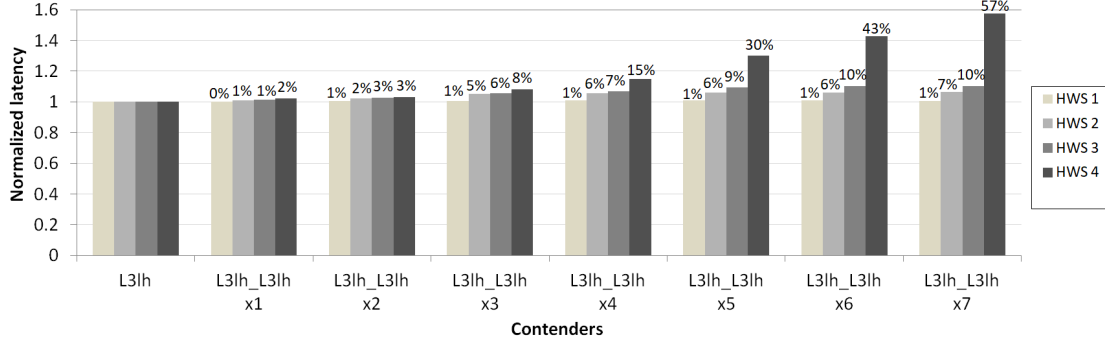
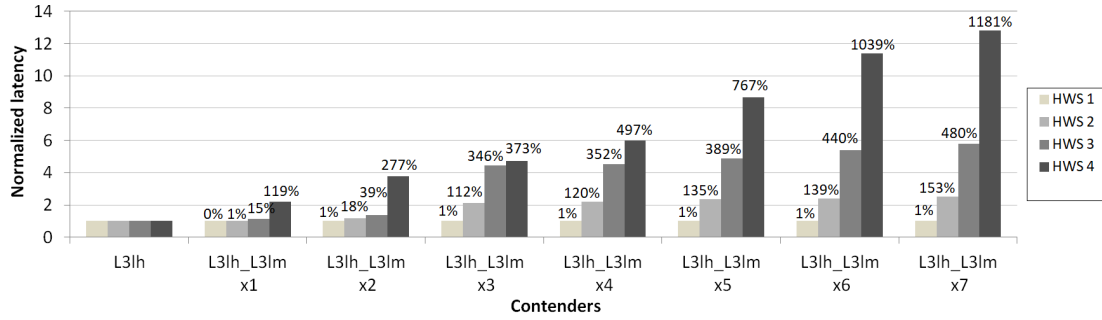

 (a) Normalized latency with *L3lh* contenders

 (b) Normalized latency with *L3lm* contenders

Figure 8.5: L3 contention under different HWS.

addition, cores also share the same CPC bank, either partitioned or shared, there is an additional slowdown per additional contender due to CPC port contention. In the case of HWS 2 and HWS 3, where only contenders 1 to 3 hit in the same CPC as *c0* (*CPC1*) and the remaining cores access to *CPC2*, there is no big difference between the partitioned and shared CPC HWS (as all benchmarks fit in the CPC) and CoreNet and CPC port contention cause a slowdown that ranges up to 10% with all cores running at the same time. Accordingly the CPC presents access contention, in which – despite tasks use different cache parts – they impact on each others access time to the cache. This is further illustrated in Figure 8.5b.

Figure 8.5b reflects how when contenders perform CPC misses, *c0* experiences an expected significant slowdown in HWS 3 and HWS 4 where CPC space is shared (i.e., software partitions evict each others' data). However, we see that it also suffers a significant slowdown in HWS 2 despite cache space not being shared. Although it cannot be proven based only on measurements, the CPC may have some internal buffers that get filled with CPC miss requests, which delay new requests. In any case, our experiment reveals the existence of severe interference channels despite CPC partitioning.

The memory controller and main memory are other well-known sources of contention. HWS 4 results reveal that while tasks are assigned different address spaces, a task can suffer a high interference from another task as those address spaces can be mapped to the same DRAM bank, device and rank. This interference is not just the time the other task is using the DRAM, but the other task can change the state of the DRAM (e.g. the open page, the direction of the bus, etc). This can cause very high delays in the task under analysis. Finally, the coherence method in place can also impact tasks' execution time by causing implicit accesses to shared resources like the interconnect or shared caches.

In addition, some interference channels may exist only under the presence of faults. Thus, assessing whether they exist is challenging until faults occur, which may be too late to take any action. For instance, some logic may exist to drive memory accesses to either *CPC1* or *CPC2*. Further, such logic, despite shared, may not have any impact in performance if the CoreNet serializes accesses. However, a fault in such logic might affect any subset of the cores in non-obvious ways and therefore, fault containment at the RRP can not be claimed without a thorough analysis of processor internals.

8.4 Summary

The CRTES development ecosystem is undergoing massive changes and as a result, their development processes are being continually refined and certification needs are continually reinterpreted to contemplate novel design paradigms. However, safety standards are relatively unchanged. CAST-32A provides a step forward in terms of guidance for the certification of systems incorporating multicore processors. CAST-32A guideline, as many safety-related standards, is abstract enough to have a broad application. However, a practical application of CAST-32A is challenging. In this chapter, we have analyzed the difficulties to apply CAST-32A to real processor designs qualitatively. Then, through a particular case study, the P4080 processor, we practically and quantitatively assess those difficulties.

Our work shows that appropriate hardware configurations and smart experimentation can reduce the degree of uncertainty. However, for the studied processor, the uncertainty can be neither removed nor deemed as irrelevant due to the non-obvious interactions of tasks at hardware level. It turns out that even the most conservative partitioning measures can be subject to execution time variations in complex COTS architectures due to shared resources and hidden undocumented features. As a consequence, this requires an in-depth analysis of the architecture which, in many COTS platforms, due to the limited documentation, can only be provided by the chip vendor. We also show that some knowledge can be gained by executing experiments on the real platform, but quantitative evidence can only

mitigate uncertainty to some extent, thus leaving several open questions to fully adhere to CAST-32A.

In summary, CAST-32A recommends building a privileged relationship with the processor manufacturer for ensuring that all relevant information is known. CAST-32A reviews a precise list of objectives but one is expected to extrapolate to the specifics of any processor under consideration. In conclusion, CAST-32A is more about ensuring that problems are sufficiently studied than about how they can be solved. This is not unexpected as it is a common pattern that certification guidance should not prescribe solutions but ensure the quality of any solution.

Chapter 9

Conclusions and Future Work

Critical Real-Time Embedded System (CRTES) designers have a constant pressure to provide more software functionalities that require increased performance capabilities in order to stay competitive in the market. This, in turn, has caused a transition towards integrated approaches where several applications run together on a single hardware platform to reduce the overall system cost, size, weight and power consumption. The computing performance needs and advance software functionalities can only be met with advanced high-performance processor designs that however carry a dramatic increase in hardware and software complexity. As a result, emerging systems, like multicores, are much more advanced than the processor designs traditionally used in this kind of critical domains and their complexity results in many challenges that imperils their industrial adoption. Motivated by this challenge, the aim of this Thesis is to support the industrial viability of integrated mixed-criticality multicore systems, with especial emphasis on safety certification and timing predictability of multicore architectures. In this chapter we summarize the main contributions of the research and its impact, and we present the open areas for future research.

9.1 Summary of Contributions

This Thesis contributes to the state-of-the-art of current and future CRTES in the adoption of multicore architectures by formalizing the foundations for their certification. We achieve this goal by defining certification-amenable mixed-criticality integration design principles, with special emphasis on the Measurement-Based Probabilistic Timing Analysis (MBPTA) technique, which has been shown to provide several advantages as a means to improve the timing predictability of complex systems. In summary, the contributions presented in this Thesis are classified in three themes: (i) mixed-criticality certification upon multicores; (ii) MBPTA

safety implications; and (iii) practical multicore certification compliance.

In particular, we contribute to the certification of mixed-criticality systems including multicore architectures as follows:

- The first contribution of this Thesis sets the key certification aspects for integrating mixed-criticality applications on multicore platforms. We present a certification argument that defines IEC 61508 based safety measures at different levels of abstraction: (i) conceptual reference safety ECU incorporating a multicore processor and safety mechanisms, (ii) safety analysis and techniques at software application (partition) level, (iii) separation mechanisms, and (iv) functional safety management for the development process. In fact, we show that IEC 61508 principles can be adapted to multicores by taking current solutions as a baseline of the argumentation and tackling specific multicore challenges like the allocation of resources and the scheduling strategy such that independence guarantees are preserved.
- Secondly, we translate the generic assumptions and safety considerations of previous contribution into technical solutions by defining two alternative safety concepts for industrial railway and automotive use cases. We evaluate alternative multicore solutions: an ad-hoc solution, a complex COTS architecture and a certification-friendly COTS platform:
 - The railway safety concept relies on partitioning and virtualization on top of the ad-hoc LEON3-MC and COTS P4080 architectures.
 - The automotive safety concept defines the integration of mixed-criticality applications on the COTS AURIX TC27x multicore. Thanks to the improved determinism and the hardware protection mechanisms available in the AURIX processor, we present a solution to safely integrate mixed-criticality applications without the need of a hypervisor.

Both solutions build on MBPTA to effectively analyze the timing behavior of mixed-criticality systems and support the scheduling decisions. We claim that MBPTA provides increased scientific foundations and hence confidence with respect to the current practice of applying an engineering margin to determine WCET estimates. Safety concepts have been reviewed and assessed by a certification body that judged the assumptions and analysis considered at this design stage as suitable for their intended purpose of evaluating the mixed-criticality integration and inclusion of the MBPTA technology in the CRTES domain.

MBPTA's industry-readiness is further pursued by evaluating how MBPTA fits the prescriptions of current functional safety standards.

- The first contribution in this direction presents a quantitative approach to assess the likelihood of execution-time exceedance events with respect to the risk reduction requirements of the safety function, based on the procedure defined for random hardware faults in safety standards. To this end, we rely on MBPTA and analyze the impact of required hardware and software modifications in the context of the IEC 61508 safety development process.
- The focus of next contribution is on an essential hardware element for achieving the required random properties on hardware, the PRNG. We adapt an existing LFSR PRNG design to a MBPTA-compliant multicore architecture. In particular, we propose a low-area, low-power design that meets IEC 61508 SIL 3 safety requirements. The presented solution guarantees that time randomized resources present same probabilistic distribution at runtime and that hence, the malfunction of the PRNG does not compromise system's safe operation or availability by invalidating the WCET estimates derived from the application of MBPTA.

The last contribution of this Thesis assesses the practical compliance of current COTS design trends to multicore certification guidance.

- We provide an analysis of CAST-32A guidance report that strives for the certification of systems integrating multicore architectures. CAST-32A is to our knowledge the only explicit certification guidance formalized for multicore solutions at the time of writing. We show the main potential stumbling blocks we have found to reach CAST-32A goals in complex architectures using a specific COTS multicore architecture, the P4080. In particular, as CAST-32A is written in the scope of the avionics domain, its requirements are more restrictive than other industrial domains. For instance, the basis of CAST-32A argumentation relies on robust partitioning, or separation, which is the central concept to achieve fault containment and prevent interferences among the applications. We reveal the difficulties to a priori determine whether a complex COTS multicore platform such as the P4080 can provide such robust partitioning as its (often undocumented) internals may violate separation at microscopic level. COTS architectures' compliance to CAST-32A hence require that processor manufacturers provide all relevant information of the processor internals.

9.2 Impact

The safety certification of mixed-criticality multicore solutions is certainly one of the main challenges and a real business-oriented need in order to enable the exploitation of research results, in this case MBPTA, for a number of reasons:

- Certification has become a requirement. Not only coming from the legislation that requires a safety certified product, but also a requirement often directly specified by the customer.
- Certification is an economic need. It complements the cost-competitive replacement of mechanical parts or manually operated products by (safety) functionality performed by programmable electronics providing legal protection to the high cost associated to their failure.
- Certification is a competitive parameter. It could be related to additional functionalities or benefits included in a new safety related product within a new market niche where fewer providers offer this kind of solutions.

Accordingly, the outcomes of this Thesis – that has been done in direct contact with industry – have served to bring multicore technology, and particularly MBPTA, closer to the CRTES industry. The strategy followed to assess certification arguments, based on industrial safety concept definition and review, served to disseminate the work and associated novel technologies to a certification body. Their concept approval is a step forward in the acceptance of the presented techniques. In addition, the collaborations in the scope of PROXIMA FP7 project lead to interesting interactions with certification experts from influential avionics and space companies (Airbus and Airbus Defence and Space). Furthermore, the evaluation of MBPTA technology in an industrial railway case study involved the integration of industrial technology solutions (SYSGO's PikeOS RTOS, RAPITA Timing analysis toolset) and resulted in a number of collaborations and joint publications listed in the Introduction (Section 1.5). These include, but are not limited to, the evaluation of a credit-based arbitration policy, the extended path coverage technique and an analysis of the instrumentation overhead in the case study.

In the same line, the membership of PROXIMA in the European Mixed-Criticality Cluster (<http://mixedcriticalityforum.org>) derived in joint activities and publications with DREAMS ([DREAMS](#)) FP7 research project. PROXIMA and DREAMS projects share the same key challenges that the development and certification of mixed-criticality multicore systems involve, where the approach targeted by DREAMS includes the definition of modular safety cases composed of a large number of 'building blocks'. In addition, other work in progress is performed in the context of SAFEPOWER H2020 project ([SAFEPOWER](#)) and other PhD students, where the safety argumentation of this Thesis is used as a baseline to define a safety concept that also considers power efficiency and security aspects.

9.3 Future Work

The results and foundations of this dissertation can be extended in various directions, leading to the identification of future research paths.

- As a short-term objective, the research work of this dissertation could be extended to the certification constraints of other domains. The effort required by this porting to different application areas covered by the IEC 61508 umbrella is comparatively simple as the standards share the same philosophy and core requirements. However, IEC 61508 does not cover all engineering domains, like space or avionics, that have also shown a big interest in multicore architectures and MBPTA technology. A promising future work should therefore consider the constraints of avionics certification standards like DO-178 ([DO178](#)) and DO-256 ([DO297](#)) or software qualification needs of the space domain (e.g., ECSS-E-ST-40C/ECSS-Q-ST-80C ([ECSS](#))). In addition, this also involves considering the limits of fail operational systems, common in the avionics domain, where timing correctness is still more crucial because the system does not dispose of a safe state to reach in case of failure and needs to remain operational.
- The relentless adoption of novel value-added functionalities does not only increase the computational demands and system complexity, but also the risk of security attacks on the system overtakes a new dimension. Security can also have an influence on system safety by, for instance, the misuse of offered services or an unauthorized access and modification of the (safety-related) software. As a result, in future CRTES, safety and security shall go hand by hand and safety functions have to be secured against attacks. Accordingly, as a mid-term objective, the certification arguments presented on this dissertation could be aligned with security requirements to systematically address both concerns consistently. For example, the proposed partition level FMEAs shall be extended to include an analysis of security threats complemented by the definition of the associated protection mechanisms.
- In the same research line, the impact of MBPTA technology to security standards could be evaluated in an analogous way to the safety certification analysis conducted in this research work. While randomization may often be seen as counter-intuitive for safety purposes, it could provide numerous advantages for security.
- Another key non-functional property to be considered is energy efficiency. A well known property of multicore architectures is in fact their potential to reduce power consumption. Though, safety or average performance are

often prioritized over energy efficiency. As for security, low power consumption solutions should be considered in the scope of safety critical systems in multicore architectures and their impact on timing.

- Finally, in the long term, as the core count in new generation processors continues to increase, further research shall consider the potential adoption by the CRTES industry of different design paradigms, like many-core architectures or MPSoCs that combine multiple processors with discrete graphics processing units (e.g., NVIDIA DRIVE PX ([NVIDIA](#)) that already pursues automotive safety certification) for high-end applications like autonomous driving. Of course such approaches massively increase hardware complexity and will involve new certification and timing predictability challenges that should be addressed in order to foster their industrial application. We strongly believe that multicores are the first fundamental step for gaining experience towards increased parallelization in CRTES domains.

References

- [AAA+17] I. Agirre, J. Abella, M. Azkarate-Askasua, and F. J. Cazorla, “On the Tailoring of CAST-32A Certification Guidance to Real COTS Multicore Architectures,” in *12th IEEE International Symposium on Industrial Embedded Systems (SIES-17)*, 2017. [10](#)
- [AAH+15] I. Agirre, M. Azkarate-askasua, C. Hernandez, J. Abella, J. Perez, T. Vardanega, and F. J. Cazorla, “IEC-61508 SIL 3 Compliant Pseudo-Random Number Generators for Probabilistic Timing Analysis,” in *2015 Euromicro Conference on Digital System Design*, 2015, pp. 677–684. [9](#)
- [AAL+15] I. Agirre, M. Azkarate-Askasua, A. Larrucea, J. Perez, T. Vardanega, and F. J. Cazorla, “A Safety Concept for a Railway Mixed-Criticality Embedded System Based on Multicore Partitioning,” in *2015 IEEE International Conference Dependable, Autonomic and Secure Computing*, 2015, pp. 1780–1787. [8](#)
- [AAL+16] I. Agirre, M. Azkarate-askasua, A. Larrucea, J. Perez, T. Vardanega, and F. J. Cazorla, “Automotive Safety Concept Definition for Mixed-Criticality Integration on a COTS Multicore,” in *Computer Safety, Reliability, and Security: SAFECOMP 2016, SASSUR Workshop, Trondheim, Norway, September 20, 2016, Proceedings*. Springer International Publishing, 2016, pp. 273–285. [8](#)
- [AAS+16] I. Agirre, M. Azkarate-askasua, I. Saenz, F. Cros, V. Jegu, C. Maxim, B. Lesage, L. Kosmidis, E. Mezzetti, and D. Griffin, “D4.8 Final case studies experiments results,” 2016, *PROXIMA project deliverable*. [42](#), [105](#)
- [AMI+11] M. Azkarate-askasua, I. Martinez, X. Iturbe, and R. Obermaisser, “Suitability of Hypervisor and MPSoC Architectures for the Execution Environment of an Integrated Embedded System,” in

- Object/Component/Service-Oriented Real-Time Distributed Computing Workshops (ISORCW)*, 2011 14th IEEE International Symposium on, 2011, pp. 19–24. 3
- [AbsInt] AbsInt, “AbsInt, Relation to Safety Standards.” [Online]. Available: <http://www.absint.com/qualification/safety.htm> 34
- [ACQ+13] J. Abella, F. Cazorla, E. Quiñones, and T. Vardanega, “Measurement-based probabilistic timing analysis and i.i.d property,” 2013, White Paper. [Online]. Available: <http://www.proartis-project.eu/publications/MBPTA-white-paper> 46
- [ARINC653] Aeronautical Radio Inc, “ARINC-653: Avionics application Software standard interface part 1 - Required Services,” 2010. 3, 28
- [AHP+14] J. Abella, D. Hardy, I. Puaut, E. Quiñones, and F. J. Cazorla, “On the Comparison of Deterministic and Probabilistic WCET Estimation Techniques,” in *2014 26th Euromicro Conference on Real-Time Systems*, 2014, pp. 266–275. 23, 24
- [AHQ+15] J. Abella, C. Hernandez, E. Quiñones, F. J. Cazorla, P. R. Conmy, M. Azkarate-Askasua, J. Perez, E. Mezzetti, and T. Vardanega, “WCET analysis methods: Pitfalls and challenges on their trustworthiness,” in *Industrial Embedded Systems (SIES)*, 2015 10th IEEE International Symposium on, 2015, pp. 1–10. xi, 6, 7, 24, 32
- [Alf96] P. Alfke, *Efficient Shift Registers, LFSR Counters, and Long Pseudo-Random Sequence Generators*, Xilinx, 1996. 136, 137
- [ALR+04] A. Avizienis, J. C. Laprie, B. Randell, and C. Landwehr, “Basic concepts and taxonomy of dependable and secure computing,” *IEEE Transactions on Dependable and Secure Computing*, vol. 1, no. 1, pp. 11–33, 2004. 16, 18
- [APC+17] J. Abella, M. Padilla, J. D. Castillo, and F. J. Cazorla, “Measurement-Based Worst-Case Execution Time Estimation Using the Coefficient of Variation,” *ACM Transactions on Design Automation of Electronic Systems (TODAES)*, vol. 22, no. 4, pp. 72:1–72:29, Jun. 2017. 46
- [AMBA] ARM Ltd., “AMBA Open Specifications,” <http://www.arm.com/products/system-ip/amba/amba-open-specifications.php>. 120

REFERENCES

- [AUTOSAR] “AUTOSAR: Automotive Open System Architecture.” [Online]. Available: www.autosar.org 3, 28
- [BCD+15] S. K. Baruah, L. Cucu-Grosjean, R. I. Davis, and C. Maiza, “Mixed Criticality on Multicore/Manycore Platforms (Dagstuhl Seminar 15121),” *Dagstuhl Reports*, vol. 5, no. 3, pp. 84–142, 2015. 4
- [BCS+16] A. Blin, C. Courtaud, J. Sopena, J. Lawall, and G. Muller, “Maximizing Parallelism without Exploding Deadlines in a Mixed Criticality Embedded System,” in *2016 28th Euromicro Conference on Real-Time Systems (ECRTS)*, 2016, pp. 109–119. 4
- [BDN+16] M. Becker, D. Dasari, B. Nolic, B. Akesson, V. Nlis, and T. Nolte, “Contention-Free Execution of Automotive Applications on a Clustered Many-Core Platform,” in *2016 28th Euromicro Conference on Real-Time Systems (ECRTS)*, 2016, pp. 14–24. 32
- [BGG+14] J. Bin, S. Girbal, D. Gracia Prez, A. Grasset, and A. Merigot, “Studying co-running avionic real-time applications on multi-core COTS architectures,” in *Embedded Real Time Software and Systems (ERTS14)*, 2014. 33, 159
- [BGGM14] J. Bin, S. Girbal, D. Gracia Prez, and A. Merigot, “Using monitors to predict co-running safety-critical hard real-time benchmark behavior,” in *International Conference of Information and Communication Technology for Embedded Systems*, 2014. 33
- [BK14] H. Brock and J. Kalmbach, “AUTOSAR goes Multi-core the safe way,” Vector Informatik GmbH, Tech. Rep., July 2014. 31, 47
- [Bos13] E. Bost, “Hardware Support for Robust Partitioning in Freescale QorIQ Multicore SoCs (P4080 and derivatives), White Paper ,” Freescale Semiconductor, Tech. Rep., 2013. 31, 159
- [Bou15] J.-L. Boulanger, *CENELEC 50128 and IEC 62279 Standards*. Wiley-ISTE, 2015, ISBN: 978-1-848-21634-1. xiii, 18
- [But12] D. Buttle, “Real-Time in the Prime-Time - (Keynote talk),” in *24th Euromicro Conference on Real-Time Systems (ECRTS-12)*, GmbH, ETAS, Ed., 2012. 2, 4
- [CAA+16] F. J. Cazorla, J. Abella, J. Andersson, T. Vardanega, F. Vatrinet, I. Bate, I. Broster, M. Azkarate-Askasua, F. Wartel, L. Cucu,

- F. Cros, G. Farrall, A. Gogonel, A. Gianarro, B. Triquet, C. Hernandez, C. Lo, C. Maxim, D. Morales, E. Quiñones, E. Mezzetti, L. Kosmidis, I. Agirre, M. Fernandez, M. Slijepcevic, P. Conmy, and W. Talaboulma, “PROXIMA: Improving Measurement-Based Timing Analysis through Randomisation and Probabilistic Analysis,” in *Proceedings - 19th Euromicro Conference on Digital System Design, DSD 2016*, 2016, pp. 276–285. 7, 25, 33
- [CCM+14] E. Carrascosa, J. Coronel, M. Masmano, P. Balbastre, and A. Crespo, “XtratuM hypervisor redesign for LEON4 multicore processor,” *SIGBED Rev.*, vol. 11, no. 2, pp. 27–31, 2014. 29
- [CAST-32] Certification Authorities Software Team, “Multi-core Processors - Position Paper,” CAST-32, Tech. Rep., May 2014. 31
- [CAST-32A] —, “Multi-core Processors - Position Paper,” CAST-32A, Tech. Rep., November 2016. xiii, 10, 28, 31, 47, 151, 153
- [CSH+12] L. Cucu-Grosjean, L. Santinelli, M. Houston, C. Lo, T. Vardanega, L. Kosmidis, J. Abella, E. Mezzetti, E. Quiñones, and F. J. Cazorla, “Measurement-Based Probabilistic Timing Analysis for Multi-path Programs,” in *2012 24th Euromicro Conference on Real-Time Systems*, 2012, pp. 91–101. 7, 25, 33, 46, 133, 148
- [Cha09] R. N. Charette, “This Car Runs on Code,” In IEEE Spectrum online, Tech. Rep., 2009. [Online]. Available: http://www.real-programmer.com/interesting_things/IEEE%20SpectrumThisCarRunsOnCode.pdf 4
- [KVC+14] N. G. Chetan Kumar, S. Vyas, R. K. Cytron, C. D. Gill, J. Zambreno, and P. H. Jones, “Cache design for mixed criticality real-time systems,” in *Computer Design (ICCD), 2014 32nd IEEE International Conference on*, 2014, pp. 513–516. 32
- [CKW+17] F. Cros, L. Kosmidis, F. Wartel, D. Morales, J. Abella, I. Broster, and F. J. Cazorla, “Dynamic software randomisation: Lessons learned from an aerospace case study,” in *Design, Automation Test in Europe Conference Exhibition (DATE), 2017*, 2017, pp. 103–108. 45
- [COBHAM] COBHAM, “LEON3 Processor. Probabilistic platform,” <http://www.gaisler.com/index.php/products/processors/leon3>. 26, 119

REFERENCES

- [COB16] Cobham Gaisler, “LEON3 SPARC V8 System-on-Chip, User’s Manual,” Cobham Gaisler AB, Tech. Rep. 2.1, 2016, EC FP7 grant agreement no 611085. 39, 67, 75
- [CONCERTO] “CONCERTO - Guaranteed Component Assembly with Round Trip Analysis for Energy Efficient High-integrity Multicore Systems.” [Online]. Available: <http://www.concerto-project.org/> 42
- [Cor12] E. J. Correia, “Sorry, Moore’s Law: Multicore Is The New Game In Town,” 2012. [Online]. Available: <http://www.crn.com/news/components-peripherals/240003030/sorry-moores-law-multicore-is-the-new-game-in-town.htm> 4
- [CQV+13] F. J. Cazorla, E. Quiñones, T. Vardanega, L. Cucu, B. Triquet, G. Bernat, E. Berger, J. Abella, F. Wartel, M. Houston, L. Santinelli, L. Kosmidis, C. Lo, and D. Maxim, “PROARTIS: Probabilistically Analyzable Real-Time Systems,” *ACM Trans. Embed. Comput. Syst.*, vol. 12, no. 2s, pp. 1–26, 2013. 7, 33
- [CRV+09] V. Cakarević, P. Radojković, J. Verdú, A. Pajuelo, F. J. Cazorla, M. Nemirovsky, and M. Valero, “Characterizing the resource-sharing levels in the UltraSPARC T2 processor,” in *Proceedings of the 42nd Annual IEEE/ACM International Symposium on Microarchitecture*. ACM, 2009, pp. 481–492. 43
- [CVH08] M. Colnatic, D. Verber, and W. A. Halang, *Distributed Embedded Control Systems: Improving Dependability with Coherent Design*. Springer Publishing Company, Incorporated, 2008, ISBN: 1848000510, 9781848000513. 1, 6
- [CVQ+13] F. J. Cazorla, T. Vardanega, E. Quiñones, and J. Abella, “Upper-bounding program execution time with extreme value theory,” in *13th International Workshop on Worst-Case Execution Time Analysis (WCET 2013)*, 2013. 46
- [CWK+15] M. Chisholm, B. C. Ward, N. Kim, and J. H. Anderson, “Cache Sharing and Isolation Tradeoffs in Multicore Mixed-Criticality Systems,” in *2015 IEEE Real-Time Systems Symposium*, 2015, pp. 305–316. 32
- [DFK+17] E. Díaz, M. Fernández, L. Kosmidis, E. Mezzetti, C. Hernandez, J. Abella, and F. J. Cazorla, *MC2: Multicore and Cache Analysis via Deterministic and Probabilistic Jitter Bounding*. Springer International Publishing, 2017, pp. 102–118. 84, 104, 121

- [DGG09] E. Dustin, T. Garrett, and B. Gauf, *Implementing Automated Software Testing: How to Save Time and Lower Costs While Raising Quality*. Addison-Wesley Professional, 2009, ISBN: 0321580516, 9780321580511. 6
- [DREAMS] “DREAMS - Distributed REal-time Architecture for Mixed Criticality Systems.” [Online]. Available: <https://www.uni-siegen.de/dreams/> 172
- [EASA12] EASA, “Certification Memorandum - Development Assurance of Airborne Electronic Hardware ,” Tech. Rep., 2012. 31
- [ECSS] “European Cooperation for Space Standardization (ECSS).” [Online]. Available: <http://ecss.nl/> 173
- [EN50126] EN50126, “EN50126:2012 - railway applications: The specification and demonstration of dependability, reliability, availability, maintainability and safety (RAMS),” 2012. 19, 68
- [ERTMS] European Railway Agency (ERA), “ERTMS - Set of specifications (ETCS baseline 3 and GSM-R baseline 0),” 2014. [Online]. Available: <http://www.era.europa.eu/Core-Activities/ERTMS/Pages/Set-of-specifications-2.aspx> 40, 69
- [FAQ+14] G. Fernandez, J. Abella, E. Quiñones, C. Rochange, T. Vardanega, and F. J. Cazorla, “Contention in Multicore Hardware Shared Resources: Understanding of the State of the Art,” in *14th International Workshop on Worst-Case Execution Time Analysis*, 2014, p. 3142. 32
- [Far14] G. Farrall, “Safety with Embedded Multicores,” in *5th Multicore Challenge*, 2014, Microcontrollers Infineon UK. 2, 5
- [XtratuM] fentISS and Universitat Politctica de Valncia (UPV), “The XtratuM Hypervisor.” [Online]. Available: <http://www.xtratum.org/> 29
- [FGQ+12] M. Fernández, R. Gioiosa, E. Quiñones, L. Fossati, M. Zulianello, and F. J. Cazorla, “Assessing the Suitability of the NGMP Multicore Processor in the Space Domain,” in *Proceedings of the Tenth ACM International Conference on Embedded Software*, ser. EM-SOFT ’12. NY, USA: ACM, 2012, pp. 175–184. 33, 43, 159
- [Fis14] S. Fisher, “Certifying Applications in a Multi-Core Environment: The World’s First Multi-Core Certification to SIL 4. White Paper,” SYSGO AG, Tech. Rep., 2014. 29, 30

REFERENCES

- [FJAQ+15] G. Fernandez, J. Jalle, J. Abella, E. Quiñones, T. Vardanega, and F. J. Cazorla, “Resource usage templates and signatures for COTS multicore processors,” in *2015 52nd ACM/EDAC/IEEE Design Automation Conference (DAC)*, 2015, pp. 1–6. [156](#), [158](#)
- [FJA+15] —, “Increasing confidence on measurement-based contention bounds for real-time round-robin buses,” in *Proceedings of the 52nd Annual Design Automation Conference*. 2744858: ACM, 2015, pp. 1–6. [33](#)
- [Free08] Freescale, “Freescale’s Embedded Hypervisor for QorIQ P4 Series Communications Platform,” Freescale Semiconductor Inc., Tech. Rep., 2008. [31](#)
- [Free09] —, “Hardware and Software Assists in Virtualization, White paper,” Freescale Semiconductor Inc., Tech. Rep., 2009. [Online]. Available: http://cache.freescale.com/files/32bit/doc/white_paper/P4080VRTASTWP.pdf [31](#)
- [FSD+13] G. Farrall, C. Stellwag, J. Diemer, and R. Ernst, “Hardware and Software Support for Mixed-Criticality Multicore Systems,” in *Workshop on Industry-Driven Approaches for Cost-effective Certification of Safety-Critical, Mixed-Criticality Systems (WICERT 2013)*, 2013. [30](#)
- [GAC+13] K. Goossens, A. Azevedo, K. Chandrasekar, M. D. Gomony, S. Goossens, M. Koedam, Y. Li, D. Mirzoyan, A. Molnos, A. B. Nejad, A. Nelson, and S. Sinha, “Virtual execution platforms for mixed-time-criticality systems: the CompSOC architecture and design flow,” *SIGBED Rev.*, vol. 10, no. 3, pp. 23–34, 2013. [30](#)
- [GB13] P. Graydon and I. Bate, “Realistic Safety Cases for the Timing of Systems,” *The computer journal*, vol. 57, no. 5, pp. 759–774, March 22 2013. [34](#)
- [GK15] D. Geiger and B. Koppenhöfer, “Integration of Mixed Criticality Systems on MultiCores: Limitations, Challenges and Way ahead for Avionics,” 2015. [Online]. Available: http://events.fortiss.org/fileadmin/uploads/events/Geiger_MultiCoreForAvionics_AirbusDS.pdf [31](#), [32](#)
- [GSN11] GSN working group, “GSN Community Standard,” Tech. Rep., 2011. [Online]. Available: <http://www.goalstructuringnotation.info/> [xi](#), [48](#)

-
- [GW09] B. Guiot, P. Winter, and International Union of Railways, *Compendium on ERTMS: European Rail Traffic Management System*. Eurailpress, 2009, ISBN: 9783777103969. 40
- [ZQ12] Z. Gu and Q. Zhao, “A State-of-the-Art Survey on Real-Time Issues in Embedded Systems Virtualization,” *Journal of Software Engineering and Applications*, vol. Vol. 5 No. 4, pp. 277–290, 2012. 29
- [HAC+17] C. Hernández, J. Abella, F. J. Cazorla, A. Bardizbanyan, J. Andersson, F. Cros, and F. Wartel, “Design and Implementation of a Time Predictable Processor: Evaluation With a Space Case Study,” in *29th Euromicro Conference on Real-Time Systems (ECRTS 2017)*, vol. 76. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik, 2017, pp. 16:1–16:23. 26, 44, 119
- [HAG+16] C. Hernandez, J. Abella, A. Gianarro, J. Andersson, and F. J. Cazorla, “Random Modulo: A new processor cache design for real-time critical systems,” in *2016 53rd ACM/EDAC/IEEE Design Automation Conference (DAC)*, 2016, pp. 1–6. 44
- [HGB+09] A. Hansson, K. Goossens, M. Bekooij, and J. Huiskens, “CoMPSoC: A template for composable and predictable multi-processor system on chips,” *ACM Trans. Des. Autom. Electron. Syst.*, vol. 14, no. 1, pp. 1–24, 2009. 30
- [Huy12] P. Huyck, “ARINC 653 and multi-core microprocessors; Considerations and potential impacts,” in *2012 IEEE/AIAA 31st Digital Avionics Systems Conference (DASC)*, 2012, pp. 6B4–1–6B4–7. 31
- [IEC61508] IEC, “Functional safety of Electrical/Electronic/Programmable Electronic safety-related systems (Second edition),” April 2010. xi, xiii, 2, 16, 17, 19, 28, 68, 117
- [AURIX] Infineon, “AURIX Family - TC27x.” [Online]. Available: <http://www.infineon.com/cms/en/product/microcontroller/32-bit-tricore-tm-microcontroller/channel.html?channel=ff80808112ab681d0112ab6b64b50805> 5, 20, 30, 39, 67, 95
- [Inf14] Infineon Technologies AG, “AURIX TC27x - User’s Manual V1.4.1,” Tech. Rep., 2014. 96
- [XEON] Intel, “XEON E7 Intel Processor Family.” [Online]. Available: <http://www.intel.es/content/www/es/es/products/processors/xeon/e7-processors.html> 21

REFERENCES

- [ISO-11898] International Organization for Standardization, “ISO-11898-4:2004 Road vehicles - Controller area network (CAN) - Part 4: Time-triggered communication,” 2004. [92](#)
- [Int09] Intel, “Applying multi-core and virtualization to industrial and safety-related applications, white paper,” Tech. Rep., 2009. [31](#)
- [ISO26262] International Organization for Standardization, “ISO/DIS 26262 Road Vehicles – Functional Safety,” 2009. [19](#), [28](#), [68](#)
- [Iti07] J.-B. ITIER, “A380 Integrated Modular Avionics. The history, objectives and challenges of the deployment of IMA on A380,” in *ARTIST2 Workshop on Integrated Modular Avionics*, Airbus, Ed., 2007. [3](#)
- [JAQ+14] J. Jalle, J. Abella, E. Quiñones, L. Fossati, M. Zulianello, and F. J. Cazorla, “AHRB: A high-performance time-composable AMBA AHB bus,” in *2014 IEEE 19th Real-Time and Embedded Technology and Applications Symposium (RTAS)*, 2014, pp. 225–236. [155](#)
- [HHM09] Jeffery Hansen and Scott Hissam and Gabriel A. Moreno, “Statistical-Based WCET Estimation and Validation,” in *9th International Workshop on Worst-Case Execution Time Analysis (WCET’09)*, ser. OpenAccess Series in Informatics (OASICS), vol. 10. Dagstuhl, Germany: Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik, 2009, pp. 1–11. [7](#)
- [JFA+16] J. Jalle, M. Fernandez, J. Abella, J. Andersson, M. Patte, L. Fossati, M. Zulianello, and F. J. Cazorla, “Bounding Resource Contention Interference in the Next-Generation Microprocessor (NGMP),” in *8th European Congress on Embedded Real Time Software and Systems (ERTS 2016)*, 2016. [155](#)
- [JKA+14] J. Jalle, L. Kosmidis, J. Abella, E. Quiones, and F. J. Cazorla, “Bus designs for time-probabilistic multicore processors,” in *2014 Design, Automation Test in Europe Conference Exhibition (DATE)*, March 2014, pp. 1–6. [26](#), [45](#), [139](#)
- [Rus99] R. John, “Partitioning in Avionics Architectures: Requirements, Mechanisms, and Assurance,” NASA Langley Technical Report Server, Tech. Rep., 1999. [28](#)

- [KAG+05] H. Kopetz, A. Ademaj, P. Grillinger, and K. Steinhammer, “The time-triggered Ethernet (TTE) design,” in *Eighth IEEE International Symposium on Object-Oriented Real-Time Distributed Computing (ISORC’05)*, 2005, pp. 22–33. 71
- [KAQ+13a] L. Kosmidis, J. Abella, E. Quiñones, and F. J. Cazorla, “A cache design for probabilistically analysable real-time systems,” in *2013 Design, Automation & Test in Europe Conference & Exhibition (DATE)*, 2013, pp. 513–518. 33, 44, 139
- [KAQ+13b] —, “Multi-level Unified Caches for Probabilistically Time Analysable Real-Time Systems,” in *2013 IEEE 34th Real-Time Systems Symposium*, Dec 2013, pp. 360–371. 26, 44, 139
- [KAQ+14] —, “Efficient Cache Designs for Probabilistically Analysable Real-Time Systems,” *IEEE Transactions on Computers*, vol. 63, no. 12, pp. 2998–3011, 2014. 44
- [KAW+14] L. Kosmidis, J. Abella, F. Wartel, E. Quiñones, A. Colin, and F. J. Cazorla, “PUB: Path Upper-Bounding for Measurement-Based Probabilistic Timing Analysis,” in *2014 26th Euromicro Conference on Real-Time Systems*, July 2014, pp. 276–287. 26
- [KCM+16] L. Kosmidis, D. Compagnin, D. Morales, E. Mezzetti, E. Quiñones, J. Abella, T. Vardanega, and F. J. Cazorla, “Measurement-Based Timing Analysis of the AURIX Caches,” in *16th International Workshop on Worst-Case Execution Time Analysis (WCET 2016)*, vol. 55. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik, 2016, pp. 9:1–9:11. 42, 45, 105
- [KCQ+13] L. Kosmidis, C.urtsinger, E. Quiñones, J. Abella, E. Berger, and F. J. Cazorla, “Probabilistic timing analysis on conventional cache designs,” in *2013 Design, Automation & Test in Europe Conference & Exhibition (DATE)*, 2013, pp. 603–606. 26, 33, 45, 120, 121
- [KF12] D. Kästner and C. Ferdinand, “Safety Standards and WCET Analysis Tools,” in *Embedded Real Time Software and Systems (ERTS 2012)*, 2012, Conference Proceedings. 34
- [Kin09] L. M. Kinnan, “Use of multicore processors in avionics systems and its potential impact on implementation and certification,” in *2009 IEEE/AIAA 28th Digital Avionics Systems Conference*, 2009, pp. 1.E.4–1–1.E.4–6. 31

REFERENCES

- [Kop04] H. Kopetz, “An integrated architecture for dependable embedded systems,” in *Proceedings of the 23rd IEEE International Symposium on Reliable Distributed Systems, 2004.*, 2004, pp. 160–161. [3](#), [5](#)
- [kop06] —, *On the Fault Hypothesis for a Safety-Critical Real-Time System*, ser. Lecture Notes in Computer Science. Springer Berlin Heidelberg, 2006, vol. 4147, book section 3, pp. 31–42. [55](#), [156](#)
- [Kop08] —, “The Complexity Challenge in Embedded System Design,” in *2008 11th IEEE International Symposium on Object and Component-Oriented Real-Time Distributed Computing (ISORC)*, 2008, pp. 3–12. [3](#)
- [Kop11] —, *Real-Time Systems: Design Principles for Distributed Embedded Applications*, 2nd ed. Springer Publishing Company, Incorporated, 2011. [1](#), [20](#), [22](#)
- [KP08] R. Kirner and P. Puschner, “Obstacles in Worst-Case Execution Time Analysis,” in *2008 11th IEEE International Symposium on Object and Component-Oriented Real-Time Distributed Computing (ISORC)*, 2008, pp. 333–339. [32](#)
- [KPG+14] D. Kästner, M. Pister, G. Gebhard, and C. Ferdinand, “Reliability of WCET Analysis,” in *Embedded Real Time Software and Systems (ERTS2014)*, 2014, Conference Proceedings. [34](#)
- [KQAF+14] L. Kosmidis, E. Quiñones, J. Abella, G. Farrall, F. Wartel, and F. J. Cazorla, “Containing Timing-Related Certification Cost in Automotive Systems Deploying Complex Hardware,” in *Proceedings of the 51st Annual Design Automation Conference*, ser. DAC ’14. NY, USA: ACM, 2014, pp. 22:1–22:6. [33](#), [121](#)
- [KQA+14] L. Kosmidis, E. Quiñones, J. Abella, T. Vardanega, I. Broster, and F. J. Cazorla, “Measurement-Based Probabilistic Timing Analysis and Its Impact on Processor Architecture,” in *17th Euromicro Conference on Digital System Design (DSD)*. 2680434: IEEE Computer Society, 2014, pp. 401–410. [7](#), [25](#), [26](#), [33](#)
- [KQA+16] L. Kosmidis, E. Quiñones, J. Abella, T. Vardanega, C. Hernandez, A. Gianarro, I. Broster, and F. J. Cazorla, “Fitting processor architectures for measurement-based probabilistic timing analysis,” *Microprocessors and Microsystems*, vol. 47, pp. 287–302, 2016. [7](#), [25](#), [26](#), [33](#), [44](#), [114](#), [119](#)

- [KVM+16] L. Kosmidis, R. Vargas, D. Morales, E. Quiñones, J. Abella, and F. J. Cazorla, “TASA: Toolchain-Agnostic Static Software randomisation for critical real-time systems,” in *2016 IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, 2016, pp. 1–8. 26, 33, 44, 45, 104, 114, 121
- [LPA+15] A. Larrucea, J. Perez, I. Agirre, V. Brocal, and R. Obermaisser, “A Modular Safety Case for an IEC-61508 Compliant Generic Hypervisor,” in *Digital System Design (DSD), 2015 Euromicro Conference on*, 2015, pp. 571–574. 81
- [LPS12] B. Lesage, I. Puaut, and A. Seznec, “PRETI: partitioned real-time shared cache for mixed-criticality real-time systems,” in *Proceedings of the 20th International Conference on Real-Time and Network Systems*. 2393009: ACM, 2012, pp. 171–180. 32
- [LS17] E. A. Lee and S. A. Seshia, *Introduction to Embedded Systems s - A Cyber-Physical Systems Approach*, second edition ed., 2017, ISBN: 978-0-262-53381-2. 1
- [MBJ09] N. Muralimanohar, R. Balasubramonian, and N. P. Jouppi, “CACTI 6.0: A tool to understand large caches,” *University of Utah and Hewlett Packard Laboratories, Tech. Rep*, 2009. 147
- [MFB+17] E. Mezzetti, M. Fernandez, A. Bardizbanyan, I. Agirre, J. Abella, T. Vardanega, and F. Cazorla, “EPC Enacted: Integration in an Industrial Toolbox and Use against a Railway Application,” in *2017 IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS)*, April 2017, pp. 163–174. 42
- [MRC11] M. Masmano, I. Ripoll, and A. Crespo, “XtratuM: a Hypervisor for Safety Critical Embedded Systems,” 2011. 29
- [MV11] E. Mezzetti and T. Vardanega, “On the industrial fitness of WCET analysis,” in *11th International Workshop on Worst Case Execution Time Analysis (WCET2011)*, 2011. 32
- [MZ91] G. Marsaglia and A. Zaman, “A New Class of Random Number Generators,” *The Annals of Applied Probability*, vol. 1, no. 3, pp. 462–480, 1991. [Online]. Available: <http://www.jstor.org/stable/2959748> 136

REFERENCES

- [Nel10] S. Nelson, “Automotive Market and Industry Update.” NXP, Director - Global Automotive Marketing. [Online]. Available: https://www.nxp.com/docs/en/supporting-information/WBNR_FTF10_AUT_F0747_PDF.pdf 2
- [NP12] J. Nowotsch and M. Paulitsch, “Leveraging Multi-core Computing Architectures in Avionics,” in *Dependable Computing Conference (EDCC), 2012 Ninth European*, 2012, pp. 132–143. 31, 33, 35, 164
- [NPB+14] J. Nowotsch, M. Paulitsch, D. Buhler, H. Theiling, S. Wegener, and M. Schmidt, “Multi-core Interference-Sensitive WCET Analysis Leveraging Runtime Resource Capacity Enforcement,” in *Real-Time Systems (ECRTS), 2014 26th Euromicro Conference on*, 2014, pp. 109–118. 32, 33, 156, 158, 159
- [NPH+14] J. Nowotsch, M. Paulitsch, A. Henrichsen, W. Pongratz, and A. Schacht, “Monitoring and WCET analysis in COTS multi-core-SoC-based mixed-criticality systems,” in *Design, Automation and Test in Europe Conference and Exhibition (DATE)*, 2014, pp. 1–5. 32
- [DS10] M. D. Natale and A. L. Sangiovanni-Vincentelli, “Moving From Federated to Integrated Architectures in Automotive: The Role of Standards, Methods and Tools,” *Proceedings of the IEEE*, vol. 98, no. 4, pp. 603–620, 2010. 3, 5
- [NVIDIA] “NVIDIA DRIVE PX. Scalable supercomputer for autonomous driving,” <http://www.nvidia.com/object/drive-px.html>. 174
- [MCP5xx] NXP, “Automotive and Industrial 32-bit MCUs.” [Online]. Available: https://www.nxp.com/products/microcontrollers-and-processors/power-architecture-processors/mpc5xxx-55xx-32-bit-mcus:POWER_ARCH_5XXX 5
- [NXiMX] —, “i.MX Applications Processors based on ARM Cores.” [Online]. Available: http://www.nxp.com/products/microcontrollers-and-processors/arm-processors/i-mx-applications-processors:IMX_HOME?&tid=FSH 21
- [P4080] —, “P4 series P4080 multicore processor.” [Online]. Available: <http://www.nxp.com/products/microcontrollers-and-processors/power-architecture-processors/qorik-platforms/p-series/qorik-p4080-p4040-p4081-multicore-communications-processors:P4080> 10, 20, 21, 38, 67, 75, 151, 159

- [NXPLS] —, “QorIQ Layerscape Processors Based on ARM Technology.” [Online]. Available: <http://www.nxp.com/products/microcontrollers-and-processors/arm-processors/qoriq-layerscape-arm-processors:QORIQ-ARM> 21
- [PAH+15] M. Panic, J. Abella, C. Hernandez, E. Quiñones, T. Ungerer, and F. J. Cazorla, “Enabling TDMA Arbitration in the Context of MBPTA,” in *2015 Euromicro Conference on Digital System Design*, 2015, pp. 462–469. 32
- [PDK+15] M. Paulitsch, O. M. Duarte, H. Karray, K. Mueller, D. Muench, and J. Nowotsch, “Mixed-Criticality Embedded Systems – A Balance Ensuring Partitioning and Performance,” in *2015 Euromicro Conference on Digital System Design*, 2015, pp. 453–461. 4, 32
- [PGN+14] J. Perez, D. Gonzalez, C. F. Nicolas, T. Trapman, and J. M. Garate, “A Safety Certification Strategy for IEC-61508 Compliant Industrial Mixed-Criticality Systems Based on Multicore Partitioning,” in *2014 17th Euromicro Conference on Digital System Design*, 2014, pp. 394–400. 2, 3, 4, 31, 55
- [PGT+14] J. Perez, D. Gonzalez, S. Trujillo, T. Trapman, and J. M. Garate, “A safety concept for a wind power mixed-criticality embedded system based on multicore partitioning,” in *11th International TÜV Rheinland Symposium "Functional Safety in Industrial Applications"*, 2014. 4, 55
- [Poo07] J. Poovey, *Characterization of the EEMBC Benchmark Suite*, North Carolina State University, 2007. 147
- [PQC+09] M. Paolieri, E. Quiñones, F. J. Cazorla, G. Bernat, and M. Valero, “Hardware support for WCET analysis of hard real-time multicore systems,” *SIGARCH Comput. Archit. News*, vol. 37, no. 3, pp. 57–68, 2009. 5, 32, 139, 156
- [PROXIMA] “PROXIMA: Probabilistic real-time control of mixed-criticality multicore and manycore systems.” [Online]. Available: <http://www.proxima-project.eu/> 33
- [DO297] Radio Technical Commission for Aeronautics (RTCA), “DO-297/ED-124 - Integrated Modular Avionics (IMA) Development Guidance and Certification Considerations,” 2005. 3, 28, 173

REFERENCES

- [DO178] —, “DO-178C - Software Considerations in Airborne Systems and Equipment Certification,” 2011. 28, 173
- [RAE+07] J. Rosen, A. Andrei, P. Eles, and Z. Peng, “Bus Access Optimization for Predictable Implementation of Real-Time Applications on Multiprocessor Systems-on-Chip,” in *28th IEEE International Real-Time Systems Symposium (RTSS 2007)*, 2007, pp. 49–60. 32
- [Ram07] J. W. Ramsey, “Integrated Modular Avionics: Less is More,” 2007. [Online]. Available: <http://www.aviationtoday.com/2007/02/01/integrated-modular-avionics-less-is-more/> 3, 4, 5
- [RPT178] Rapita Systems LTD, “Automating WCET analysis for DO-178B/C, White Paper,” Rapita systems LTD, Tech. Rep. [Online]. Available: www.rapitasystems.com 34
- [RPT] —, “Rapita Verification Suite (RVS),” 2016. [Online]. Available: <https://www.rapitasystems.com/> 34
- [Reb17] J. Rebel, “Infineon AURIX 32-bit microcontrollers as the basis for ADAS / Automated Driving,” in *Deutsche Bank AutoTech Conference*, 2017. 2
- [RGG+12] P. Radojković, S. Girbal, A. Grasset, E. Quiñones, S. Yehia, and F. J. Cazorla, “On the Evaluation of the Impact of Shared Resources in Multithreaded COTS Processors in Time-critical Environments,” *ACM Trans. Archit. Code Optim.*, vol. 8, no. 4, pp. 34:1–34:25, Jan. 2012. 43
- [RM14] D. Reinhardt and G. Morgan, “An embedded hypervisor for safety-relevant automotive E/E-systems,” in *Industrial Embedded Systems (SIES), 2014 9th IEEE International Symposium on*, 2014, pp. 189–198. 29
- [RSN+10] A. Rukhin, J. Soto, J. Nechvatal, and S. Vo, “A Statistical Test Suite for the Validation of Random Number Generators and Pseudo Random Number Generators for Cryptographic Applications,” US National Institute of Standards and Technology (NIST), Special Publication 800-22rev1a, 2010. 134, 135, 136, 138, 141, 146
- [RVS] “RAPITA Verification Suite.” [Online]. Available: <https://www.rapitasystems.com/products/rvs> 46

- [S4R] “Safe4RAIL: Safe architecture for Robust distributed Application Integration in roLling stock.” [Online]. Available: <https://safe4rail.eu/> 28
- [SAE01] “Guidelines and Methods for Conducting the Safety Assessment Process on Civil Airborne Systems and Equipment,” *ARP4761*, p. 331, 2001, AKA:SAE ARP 4761. 3
- [SAFEPOWER] “SAFEPOWER - Safe and secure mixed-criticality systems with low power requirements.” [Online]. Available: <http://safepower-project.eu/> 172
- [ZAV13] Z. Stephenson, J. Abella, and T. Vardanega, “Supporting industrial use of probabilistic timing analysis with explicit argumentation,” in *11th IEEE International Conference on Industrial Informatics (INDIN)*, 2013, pp. 734–740. 34
- [SEH+12] C. E. Salloum, M. Elshuber, O. Hoftberger, H. Isakovic, and A. Wasicek, “The ACROSS MPSoC – A New Generation of Multi-core Processors Designed for Safety-Critical Embedded Systems,” in *Digital System Design (DSD), 2012 15th Euromicro Conference on*, 2012, pp. 105–113. 5, 27, 30, 32
- [SMM00] J. Swingler, J. McBride, and C. Maul, “Degradation of road tested automotive connectors,” *IEEE Transactions on Components and Packaging Technologies*, vol. 23, no. 1, pp. 157–164, Mar 2000. 3
- [Ste08] W. Steiner, “TTEthernet Specification,” *TTTech D-INT-S-10-002*, 20th November 2008. 71
- [PikeOS] SYSGO, “PikeOS Hypervisor.” [Online]. Available: <https://www.sysgo.com/products/pikeos-hypervisor/> 29, 41
- [PikSIL4] SYSGO, “PikeOS Product Information: PikeOS SIL 4 certification on multi-core platform,” October 23 2013. [Online]. Available: <http://www.sysgo.com/news-events/press/press/details/article/pikeosTM-sil-4-certification-on-multi-core-platform/> 29, 47
- [SAA+14] S. Trujillo, A. Crespo, A. Alonso, and J. Prez, “MultiPARTES: Multi-core partitioning and virtualization for easing the certification of mixed-criticality systems,” *Microprocessors and Microsystems*, vol. 38, no. 8, Part B, pp. 921–932, 11// 2014. 31

REFERENCES

- [TLH97] V. Tran, L. Dar-Biau, and B. Hummel, “Component-based systems development: challenges and lessons learned,” in *Proceedings Eighth IEEE International Workshop on Software Technology and Engineering Practice incorporating Computer Aided Software Engineering*, 1997, pp. 452–462. 4
- [TICon] Texas Instruments, “Overview for Control and Automation MCUs.” [Online]. Available: <http://www.ti.com/lsds/ti/microcontrollers-16-bit-32-bit/c2000-performance/control-automation/overview.page> 21
- [TMS570] Texas Instruments (TI), “Overview for Hercules TMS570 MCUs.” [Online]. Available: http://www.ti.com/lsds/ti/microcontrollers-16-bit-32-bit/c2000_performance/safety/tms570/overview.page 5
- [VYF16] P. K. Valsan, H. Yun, and F. Farshchi, “Taming Non-Blocking Caches to Improve Isolation in Multicore Real-Time Systems,” in *2016 IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS)*, 2016, pp. 1–12. 156
- [WEE+08] R. Wilhelm, J. Engblom, A. Ermedahl, N. Holsti, S. Thesing, D. Whalley, G. Bernat, C. Ferdinand, R. Heckmann, T. Mitra, F. Mueller, I. Puaut, P. Puschner, J. Staschulat, and P. Stenström, “The worst-case execution-time problem – overview of methods and survey of tools,” *ACM Trans. Embed. Comput. Syst.*, vol. 7, no. 3, pp. 1–53, 2008. xi, 6, 22, 23
- [WINDRIVER] Wind River, “Wind River Virtualization,” 2011. [Online]. Available: <http://www.windriver.com/products/operating-environments/virtualization/> 29
- [Win12] J. Windsor, “Integrated Modular Avionics for Space (IMA4Space),” in *6th ESA Workshop on Avionics, Data, Control and Software Systems (ADCSS-2012)*, 2012, european Space Agency (ESA). 3
- [VxSP15] Wind River, “Safety Profile for VxWorks, Product Overview,” Wind River, Tech. Rep., 2015. [Online]. Available: https://www.windriver.com/products/product-overviews/Safety-Profile-for-VxWorks_Product-Overview/ 29
- [Vx653] —, “VxWorks 653 3.0 Multi-Core Edition, Product Overview,” Wind River, Tech. Rep., 2015. [Online]. Available: <https://www.windriver.com/products/product-overviews/vxworks-653-product-overview-multi-core/> 29

- [VxCer15] —, “Wind River VxWorks Cert Platform, Product Overview,” Wind River, Tech. Rep., 2015. [Online]. Available: <https://www.windriver.com/products/product-overviews/vxworks-cert-product-overview/> 29
- [WKG+15] F. Wartel, L. Kosmidis, A. Gogonel, A. Baldovin, Z. Stephenson, B. Triquet, E. Quiñones, C. Lo, E. Mezzetti, I. Broster, J. Abella, L. Cucu-Grosjean, T. Vardanega, and F. J. Cazorla, “Timing analysis of an avionics case study on complex hardware/software platforms,” in *Proceedings of the 2015 Design, Automation & Test in Europe Conference & Exhibition*, 2015, pp. 397–402. 7
- [WKL+13] F. Wartel, L. Kosmidis, C. Lo, B. Triquet, E. Quiñones, J. Abella, A. Gogonel, A. Baldovin, E. Mezzetti, L. Cucu, T. Vardanega, and F. J. Cazorla, “Measurement-based probabilistic timing analysis: Lessons from an integrated-modular avionics case study,” in *2013 8th IEEE International Symposium on Industrial Embedded Systems (SIES)*, 2013, pp. 241–248. 7, 33
- [WT08] A. Wilson and T. Preysler, “Incremental certification and Integrated Modular Avionics,” in *2008 IEEE/AIAA 27th Digital Avionics Systems Conference*, 2008, pp. 1.E.3–1–1.E.3–8. 2
- [ZynqFS] Xilinx and IK4-IKERLAN, “Zynq-7000 Functional Safety Reference System: SPS Drives 2016.” [Online]. Available: <http://www.origin.xilinx-china.com/video/events/zynq-7000-functional-safety-reference-system.html> 5
- [ZMV+15] M. Ziccardi, E. Mezzetti, T. Vardanega, J. Abella, and F. J. Cazorla, “EPC: Extended Path Coverage for Measurement-Based Probabilistic Timing Analysis,” in *2015 IEEE Real-Time Systems Symposium*, Dec 2015, pp. 338–349. 26

List of Acronyms

ASIL	Automotive Safety Integrity Level	ERTMS	European Railway Traffic Management System
ATD	Analysis-time Distribution	ETCS	European Train Control System
BIST	Built-in Self-Test	EVC	European Vital Computer
CAN	Controller Area Network	EVT	Extreme Value Theory
CC	Cruise Control	FCR	Fault Containment Region
CCDF	Complementary Cumulative Distribution Function	FPGA	Field-Programmable Gate Array
COTS	Commercial Off-The-Shelf	FPU	Floating Point Unit
CPU	Central Processing Unit	FMEA	Failure Mode and Effect Analysis
CPC	CoreNet Platform Cache	FSM	Functional Safety Management
CRTES	Critical Real-Time Embedded Systems	FIT	Failures In Time
DC	Diagnostic Coverage	GSN	Goal Structuring Notation
DSR	Dynamic Software Randomization	HFT	Hardware Fault Tolerance
DTA	Deterministic Timing Analysis	HWM	High-Water Mark
ECU	Electronic Control Unit	i.i.d.	independent and identically distributed
ECCDF	Empirical Complementary Cumulative Distribution Function	IMA	Integrated Modular Avionics

LFSR	Linear Feedback Shift Register	RP	Random Placement
MBTA	Measurement-Based Timing Analysis	RR	Random Replacement
MBPTA	Measurement-Based Probabilistic Timing Analysis	RRP	Robust Resource Partitioning
MMU	Memory Management Unit	RTP	Robust Time Partitioning
MPU	Memory Protection Unit	SSR	Static Software Randomization
MPSoC	Multi-Processor System-on-a-Chip	MxCPU	Mixed-criticality Central Processing Unit
MWC	Multiply-With-Carry	SFF	Safe Failure Fraction
NoC	Network-on-Chip	SIL	Safety Integrity Level
OTD	Operation-time Distribution	SoJ	Sources of Jitter
MOET	Maximum Observed Execution Time	SoR	Source of Randomization
OS	Operating System	SPB	System Peripheral Bus
PMC	Performance Monitoring Counters	SRI	Shared Resource Interconnect
PRNG	Pseudo-Random Number Generator	STA	Static Timing Analysis
PTA	Probabilistic Timing Analysis	TASA	Toolchain-Agnostic Software rAndomization
pWCET	probabilistic WCET	TLB	Translation Lookaside Buffer
		UoA	Unit of Analysis
		WCET	Worst-Case Execution Time